CSE 8A Programming Assignment 3

Due Date: Tue. Oct 27th, 11:59 PM (PDT)

Learning goals:

- Further practice using lists and strings
- More conditional statements
- Learn how to use loops

Logistics:

All information for the following can be found on our course syllabus.

- **Pair programming** - This programming assignment can be done individually or with a partner. Make sure to read the guide on pair programming.
- **Academic integrity** - Please adhere to all academic integrity guidelines on the syllabus.

Submission:

- You will submit the following files to Gradescope:
  - **pa3.py** - Contains your code for part 1.1 (and star points, if attempted)
  - **pa3-game.py** - Contains your code for part 1.2
  - **Pa3-writeup.pdf** - Contains all written portions of the assignment
  - **Pa3-video.mp4** - Your recorded video
- You can find the template of the write-up here
- You can find a video demo of part 1.2 here
- You are also required to complete the weekly reflection
- Instructions on how to submit can be found below at submission instructions.

Part 1: Implementation (6 Points)

Part 1.1: Iterating over lists

*Save all of the following code in the provided python file named **pa3.py.***

You want to send pumpkins to some of your friends for Halloween! You look up shipping policies and find out that there are certain constraints involved when shipping packages. A pumpkin needs to weigh at least a certain amount in order to be eligible for shipping. At the same time, no pumpkin can weigh more than a certain amount due to packing restrictions.

However, you have already purchased a whole lot of pumpkins in varying sizes. You now want to find out which of these pumpkins are eligible for shipping. A pumpkin is eligible for shipping if it weighs at least as much as the minimum required weight, and not more than the maximum permitted shipping weight. You have the weights of all pumpkins in a Python list (let's call this *pumpkin_weights*).

Define a function *get_shippable_pumpkins* that takes 3 parameters: a list of numbers named *pumpkin_weights*, and two floats *min_weight*, and *max_weight*. The function should compare each value in the list with the minimum and maximum weights allowed (i.e., *min_weight* and *max_weight*), and return a list of pumpkins that are within the weight limits for shipping.

You may assume that the *min_weight* is always less than or equal to *max_weight*. In other words, the lower limit of weight will always be less than or equal to the upper limit.

Implement the following function to perform the task as described. The name of your function should match exactly as shown, including cases (all lowercase) and underscores. Our autograder on gradescope will not be able to test your function if it is not named properly.

Function Name: *get_shippable_pumpkins*
Parameter: *pumpkin_weights* - A list of floats that represents the weights of pumpkins. *min_weight* - Minimum allowed weight for a pumpkin to be shippable (inclusive) *max_weight* - Maximum allowed weight for a pumpkin to be shippable (inclusive)
Return: A list of pumpkins that are shippable.
Description: Create and return shippable pumpkins, given a list of pumpkin weights and a minimum and maximum weight allowed for shipping.
Examples: pumpkin_weights = [33.0, 36.3, 25.1, 29.4, 31.5] min_weight = 30.0 max_weight = 34.0 Returns: [33.0, 31.5] pumpkin_weights = [30.6, 21.1, 35.0] min_weight = 35.0 max_weight = 35.0 Returns: [35.0] pumpkin_weights = [16.9, 42.0, 25.5, 24.3, 32.6, 29.1, 30.3, 31.6] min_weight = 22.5 max_weight = 34.5

```
Returns: [25.5, 24.3, 32.6, 29.1, 30.3, 31.6]

pumpkin_weights = [33.0, 36.0, 35.5, 33.6, 41.2]
min_weight = 36.9
max_weight = 40.1
Returns: [ ]
```

It is always a good practice to think about edge cases while writing programs. For example, if you find no pumpkins are eligible for shipping, you should return an empty list. Similarly, if the input list is empty, you should return an empty list again. Observe that if the two limits are identical (as in the second example), we are effectively looking for an exact weight for shipping-eligible pumpkins.

You **should not** take input from the user for this. While testing your program, you may use your own test cases and pass them to the function by calling the function from the python prompt (as we do in lectures). For this part of the assignment, we only require the *get_shippable_pumpkins* function to be written.

Part 1.2: Guessing Game

*Save all of the following code in the provided python file named **pa3-game.py**.*

We are going to build our own number guessing game! The way this game works is as follows:

Using a random number generator, pick a number between 1-100 (don't worry about writing code for this - it has been provided later in this document). The player is then asked to try and guess this number. They are given a limited number of attempts to try and guess the number correctly. We allow the **player** to decide how many attempts they want at the start of the game.

Once the player starts guessing, for each incorrect guess, the player is informed whether they must "Guess higher" or "Guess lower" the next time. You must also display the number of remaining attempts after each incorrect guess. The game ends either when the player guesses the correct number, or when they run out of attempts.

You can find a video showing a sample run of how we expect the game to work here.

We do not expect you to follow any particular format to implement this game. You are free to implement the game in your own way (as long as you use Python and concepts we have learned in CSE 8A so far). The main objective is to have **fun**! The only strict requirement is that you **must** use a **while loop** for implementing the main game logic, i.e. reading the player's

guesses, checking if each guess is correct/incorrect, and displaying the appropriate messages at each step.

For this part of the assignment, write all your code in a separate file named **pa3-game.py**. We also require you to show a demo of the game in your video (more details here).

Use the following code to generate a random number between 1-100:

```
import random
secret_number = random.randint(1, 100)
```

These two lines should be written as the first two lines in your pa3-game.py file.

Star Points (optional):

*Save all of the following code in **pa3.py***
Read about star points on our course syllabus. To obtain a star point for this PA, you will be writing the following two functions in pa3.py.

Part 1.3: Statistics
(Only for star points; Not required to get full credits on this PA)

Write a function *print_statistics* that takes a list of numbers (floats) as a parameter, and computes and **prints** out the following statistics:

- The maximum element in the sequence.
- The minimum element in the sequence.
- The mean (average) of the sequence. The mean is defined as the sum of all numbers in the sequence divided by the number of elements in the sequence.
- The number of occurrences of the number zero (0 or 0.0), the number of negative numbers, and the number of positive numbers in the sequence.

Function Name: *print_statistics*
Parameter: *data* - A list of floating point numbers.
Return: None.
Description: Given a list of floats, print out the maximum and minimum elements in the list, the mean (average) of the sequence, and the number of zeros, negative numbers and positive numbers in the sequence.

Example:
Parameter: data - [90.0, 92.2, 95.1, 85.4, 92.2, 100.0]
Output:
Maximum: 100.0
Minimum: 85.4
Mean: 92.48333333333333
Number of occurrences of Zero: 0
Number of negative numbers: 0
Number of positive numbers: 6

Parameter: data - [-20.0, 10.5, 0.0, -15.1, 21.0]
Output:
Maximum: 21.0
Minimum: -20.0
Mean: -0.72
Number of occurrences of Zero: 1
Number of negative numbers: 2
Number of positive numbers: 2

You may assume that the list is non-empty (i.e., the list will contain at least one number). You can also assume that the list will only contain numbers (float).

Part 1.4: More practice with loops
(Only for star points; Not required to get full credits on this PA)

Write a function to display the following pattern:

```
*
* *
* * *
* * * *
* * * * *
```

(*n* rows)

Given a parameter *n*, write a program that prints the pattern as shown above, up to *n* rows. For example, if n=3, we print 3 rows. The pattern starts with a single asterisk (`"*"`), and each subsequent row prints one asterisk more than the previous row. The asterisks within a row are separated by a single space each. Remember to print a new line to distinguish between rows.

Function Name: *print_pattern*
Parameter: *n* - An integer that determines the number of rows of the pattern to be printed.
Return: None.

Description:
Given an integer *n*, print a pattern according to the instructions given above. This function should **NOT** return anything.

Examples:
Parameter: 3
Output:
```
*
* *
* * *
```
Returns: None

Parameter: 1
Output:
```
*
```
Returns: None

Parameter: 10
Output:
```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```
Returns: None

As before, consider edge conditions like 0, -1 etc. For all such cases, don't output anything.

HINT: You may use **string replication** to write this program.

Part 2: Write up (3 Points)

You must report how you tested your code as well as answer a few short questions in **Pa3-writeup.pdf.** A template has been provided. See submission instructions below on how to make a copy of this template to your own drive. In particular, you must provide:

A. **Report Bugs and Issues**
   Include in your report any known bugs or issues with your program.

B. Questions

Answer the following questions:

1. What are the differences between for and while loops? When would you use a for loop? When would you use a while loop?

2. Given a positive number *num*, the following code is trying to calculate and print the sum of natural numbers (i.e. positive integers greater than zero, eg. 1,2,3...) up to *num*. However, there is an error in this code. Can you identify it? How can we fix it?

```
num = 10
sum = 0
i = 1
while i <= num:
    sum = sum + i

print("Sum of natural numbers up to", num, "is:", sum)
```

3. What is the difference between indexing and slicing? When would you use one over the other?

Part 3: Video

For this part, you will create a video recording explaining the code you have written. Your video should answer the questions below. If you are working with a partner, both partners should be in the video and each partner should have some speaking point. Example of following code execution: https://youtu.be/7uw_Vi_F-dY

1. **Show a live demo of your game.** Play two scenarios: a win and a loss. Explain the code while you do so (for any one scenario). Here is an example video of how the game works. In the demo video we do not show the code but in your video, you should demo yourself playing the game AND you should also **show and explain your code**.

The following things will be checked in your video while grading:

1. The student(s) **code is clearly visible** in the video. [Hint: Increase your font size to 18
   Windows: "Options" → "Configure Idle" → Size
   Mac: "IDLE"(in top menu bar) → "Preferences" → "Fonts/Tabs" → "Size"]
2. The student(s) clearly **answer(s) the questions**.
   a. Students submitting individually answer both questions.
   b. Students submitting in pairs each answer one of the questions.
3. Video is within **time limit** (max: **2 mins**)

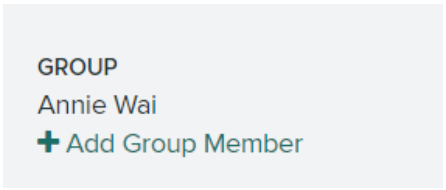
Part 4: Weekly Reflection (1 Point)

Fill out the reflection form (link here). This weekly reflection form is not optional, it counts towards 1 point of your assignment. All students have to **individually** submit their own weekly reflection regardless if you're working with a partner. Weekly reflections are not due when the PA is due. You may submit your weekly reflection for PA 3 anytime before **11:59pm on Friday Oct 30th**.

Submission Instructions

Read all instructions carefully before submitting.

- You will need to submit **pa3.py, pa3-game.py, Pa3-writeup.pdf, and Pa3-video.mp4** on Gradescope, fill out the weekly reflection, and complete the degree planning assignment.
- To copy the writeup template and export as a pdf:
    - Click here to see a copy of the assignment format.
    - Click on "File" -> "Make a copy", and you will get a local copy of this Google Doc.
    - Fill in the Google Doc, making sure you keep the headings in about the same places.
    - Once you are done, in Google Docs, click on "File"-> "Download" -> "PDF Document", which will export it to a pdf.
- To record a video on zoom:
    - Start a zoom video meeting.
    - Choose "Join with Computer Audio"
    - If you're working with a partner, invite your partner into the meeting.
    - Share your computer's screen using "Share Screen"
    - Show your code on your computer's screen
    - Click "Record" > "Record on your computer".
    - Answer the required questions.
    - Once you are done, click "stop recording" and "End meeting".
    - Save your video file on your computer and name it as Pa3-video.mp4
- Sign into Gradescope and submit all files to PA 3. You should be able to drag and drop multiple files into the upload files window. Ask a teaching staff for help if you are unsure whether you've submitted properly.
- If you are working with a partner, **only one member will need to submit the files** in Gradescope. Do not both submit the files individually to Gradescope. It will be your responsibility to ensure both members are added in Gradescope.
- To add a group member on Gradescope:
    - First submit all files to PA 3.
    - This should take you to your submissions page. Otherwise, you can view your submission by clicking on the assignment.
    - Click on **"Add Group Member"** on the top right under your name.



    - Confirm you have added your partner. You should see both you and your partner's name under "Group" in the top right after submitting.
- You may submit multiple times until the deadline. We will be grading only your latest submission. So, please make sure that your latest submission is your best version!