Assignment 4

Lists and Tuples

Date Due: Friday, October 11, 2019, 6:00pm **Total Marks: 22**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M. **Put your name, NSID and student number** at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you. Do not submit folders, zip documents, even if you think it will help.
- **Programs must be written in Python 3.** Marks will be deducted with severity if you submit code for Python 2.
- **Assignments must be submitted to Moodle.** There is a link on the course webpage that shows you how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Questions are annotated use descriptors like "easy" "moderate" and "tricky". All students should be able to obtain perfect grades on "easy" problems. Most students should obtain perfect grades on "moderate" problems. The problems marked "tricky" may require significantly more time, and only the top students should expect to get perfect grades on these. We use these annotations to help students be aware of the differences, and also to help students allocate their time wisely. Partial credit will be given as appropriate, so hand in all attempts.

Question 1 (7 points):

Purpose: To practice your ability to create and manipulate lists.

Degree of Difficulty: Easy

Pokemon are fantastic creatures that people like to catch and train. For this question, you will write a program that allows the user to keep track of the Pokemon they have caught, and then assemble a team from among the Pokemon caught that can be taken to the local Pokemon Gym for training.

Your program will have two phases. When the program first starts, it should repeatedly show a prompt to the user asking if they want to catch another Pokemon. Each time they say yes, the program should ask the user to enter the name of the Pokemon that was caught and it should be added to a list variable that holds the user's complete Pokemon collection. When they say no, the program should continue to the second phase.

Get this part of the program working before going any further. Do not move on to the second part of the program until you know that the first part works.

In the second phase, the program will display the current Pokemon collection, as well as a list of Pokemon that have been added to the user's Gym Team (initially, this second list will be empty). A prompt will then ask the user to name the Pokemon from their collection that they would like to add to the Gym Team. The Pokemon entered should be removed from the first list (the Pokemon collection) and added to the Gym Team list. This process should continue until EITHER (a) there are 6 Pokemon on the Gym Team, or (b) all of the Pokemon from the collection have already been added to the Gym Team.

Sample Run

Here is an example of how your program's console output might look. Green text was entered by the user.

```
Time to catch some Pokemon!

Which Pokemon have you caught? pikachu
Would you like to keep catching Pokemon (y/n)? y
So far you've caught:
['pikachu']
Which new Pokemon have you caught? wobuffet
Would you like to keep catching Pokemon (y/n)? y
So far you've caught:
['pikachu', 'wobuffet']
Which new Pokemon have you caught? charmander
Would you like to keep catching Pokemon (y/n)? y
So far you've caught:
['pikachu', 'wobuffet', 'charmander']
Which new Pokemon have you caught? bulbasaur
Would you like to keep catching Pokemon (y/n)? n
~~~
Time to head to the Pokemon Gym!
~~~
Pokemon in your collection:
['pikachu', 'wobuffet', 'charmander', 'bulbasaur']
Pokemon on your Gym Team:
[]
What Pokemon will you add to your team? pikachu
I CHOOSE YOU, PIKACHU!!
Pokemon in your collection:
['wobuffet', 'charmander', 'bulbasaur']
Pokemon on your Gym Team:
['pikachu']
What Pokemon will you add to your team? wobuffet
```

```
I CHOOSE YOU, WOBUFFET!!
Pokemon in your collection:
['charmander', 'bulbasaur']
Pokemon on your Gym Team:
['pikachu', 'wobuffet']
What Pokemon will you add to your team? charmander
I CHOOSE YOU, CHARMANDER!!
Pokemon in your collection:
['bulbasaur']
Pokemon on your Gym Team:
['pikachu', 'wobuffet', 'charmander']
What Pokemon will you add to your team? bulbasaur
I CHOOSE YOU, BULBASAUR!!
~~~
Your Pokemon team is ready to hit the gym!
```

You may assume that the user will enter correct input in all cases, i.e. the user will only try to add Pokemon to the Gym Team that are already in the list.

What to Hand In

Hand in your solution in a file called `a4q1.py`.

Evaluation

- **-1 mark** for missing name, NSID and student number at top of file
- 2 marks for correct behaviour of while loops
- 1 mark for console input
- 1 mark for creating a list type variable
- 1 mark for printing lists in appropriate places
- 1 mark for correctly using method append()
- 1 mark for correctly using method remove()

Question 2 (5 points):

Purpose: To practice creating and manipulating lists with list comprehensions.

Degree of Difficulty: Mostly Easy

In `a4q2-starter.py` you are given a definition of a list of lists. Each sublist contains three pieces of information on one famous prairie pirate – their pirate name, the number of sacks of assorted grains they have plundered from unsuspecting farmers in the last year, and a Boolean value indicating whether they like parrots. Create the following list comprehensions where indicated by comments in the provided file `a4q2-starter.py`. Note that you **only** need to create the appropriate list comprehensions, you do not need to print them out (though you may want to for testing purposes).

(a) Use a single list comprehension to create a **list of strings** of the pirate names who plundered more than 400 sacks of assorted grains.

(b) Use a single list comprehension to create a **list of strings** of the pirate names who don't like parrots.

(c) Suppose that the average market value of a sack of grain is $42. Use a single list comprehension to create a **list of integers** of the market values of each pirate's grain.

(d) Use a single list comprehension to create a **list of lists** where each sublist consists of a pirate's name, and the value of his/her plundered sacks of grain (calculate grain values as in part (c)), but only include those pirates who like parrots.

What to Hand In

Rename your completed `a4q2-starter.py` file to `a4q2.py` and submit it.

Evaluation

- **-1 mark** for missing name, NSID and student number at top of file
- 1 mark each for parts (a), (b), and (c).
- 2 marks for part (d).

Question 3 (10 points):

Purpose: To practice your ability to modify lists and compute values with lists

Degree of Difficulty: Moderate

Professor Oak is in trouble: his class grades are a mess! His midterm exam was too hard, and he gave out too many bonus marks to certain students. For this question, you will write a program to clean up the grades and calculate the final grade for each student in the class.

Starter File

a4q3_starter.py is a file that contains Professor Oak's student grades as a list of lists. Each sublist has exactly 6 items, representing the grades for (in order) four assignments, a midterm exam, and a final exam. All grades are out of 100. Just copy/paste this list to the top of your program. But keep in mind that your code should work even if we use a different starting list (though with the same format) to test your program.

From here, you will write three separate functions to perform three tasks, as described below.

Cleaning the Data

Write a function called `clean_grades()` that takes the list of lists of grades as a parameter. For all of the student grades, this function should perform two tasks:

- Increase the midterm exam grade for ALL students by 2. Recall that the midterm grade is the 5th item in each sublist.
- For any grade greater than 100, set the grade to be just 100 (make sure to check for this AFTER increasing the grade for the midterm exam!)

These changes should be done by modifying the input list itself. No new lists should be created and this function has no return value.

Student Final Grades

Write a function called `student_averages()` that takes the list of lists of grades as a parameter and computes a final grade for each student. The grading scheme for the class is:

- 10% for each assignment (recall there were 4 assignments)
- 20% for the midterm exam
- 40% for the final exam

The final grade for each student **must be an integer**, rounded appropriately (you can use Python's `round()` function for this). The final grades should be placed into a single list and this function must **return** that list.

Class Average

Write a function called `average()` that takes a list of integers representing the grade of each student in the class as a parameter, and **returns** the class average as a single, floating point number.

Program Output

At the very bottom of your program, call all of the functions you made above, using their return values appropriately, to print out the list containing each student's final grade, as well as the overall class average.

Sample Run

Here is an example of how your program's console output might look (the 2nd line will in fact be much longer; we are showing only the first few numbers here to save space).

```
The final grades for the class are:
[69, 76, 40, 69, 50, 58, 66, 79, 69, 86, ... ]
The overall class average is: 64.04
```

What to Hand In

Hand in your solution in a file called `a4q3.py`.

Evaluation

- **-1 mark** for missing name, NSID and student number at top of file
- 3 marks for `clean_grades()`
- 2 marks for `student_averages()`
- 2 marks for `average()`
- 1 mark for function calls and printing output
- 2 marks for docstrings for each function and appropriate comments