Assignment 3

Summary of Instructions and Overview

Note	Read the instructions carefully and follow them exactly
Assignment Weight	As outlined in the syllabus, this assignment is worth 5% of your final grade
Due Date	This assignment is due 8am on Monday, Oct 30, 2017
Important	As outlined in the syllabus, late submissions will not be accepted.
	Any Python files with syntax errors will automatically be excluded from grading. Be sure to test your code before you submit it.
	For all functions, make sure you've written good docstrings that include type contract, function description and the preconditions if any.
	This is an individual assignment. Please review the Plagiarism and Academic Integrity policy presented in the first class, i.e. read in detail pages $15-19$ of course outline. You can find that file on Brightspace under Course Information. While at it, also review Course Policies on pages 13 and 14.

The goal of this assignment is to learn and practice the concepts covered thus far: function design, lists and loops. You can make multiple submissions, but only the last submission before the deadline will be graded.

For this assignment, you only need to submit one file, called `a3_xxxxxx.py`, where you changed `xxxxxx` to your student number. (No need to submit `a3_xxxxxx.txt` as proof that you tested your function. By now we trust that you learnt and understand the need and importance for testing your functions and code in general).

For this assignment, I provided you with starter code in file called `a3_xxxxxx.py`. Begin by replacing `xxxxxx` in the file name with your student number. Then open the file. Your solution (code) for the assignment must go into that file in clearly indicated spaces. The file already has some functions completely precoded for you and some partially coded. You are not allowed to delete or comment-out any parts of the provided code. You also must follow the instructions given in comments and implied by docstrings. You are however allowed to add your own additional (helper) functions.

As always, your program must run without syntax errors. In particular, when grading your assignment, TAs will first open your file `a3_xxxxxx.py` with IDLE and press Run Module. If pressing Run Module causes any syntax error, the grade for the assignment becomes zero. Furthermore, for some of the required functions, I have provided one or more tests to test your functions with (as you can see in the video). For example, you should test function `clean_up_board` by making a call in Python shell as follows:

```
>>> clean_up_board(['A', 'B', 'C', 'B', 'B', 'B', 'C', 'B'])
```

To obtain a partial mark for these function your solutions may not necessarily give the correct answer on these tests. But if your function gives any kind of Python error when run on the tests provided in the video, that question will be marked with zero points. Finally, each function has to be documented with docstrings.

Using global variables inside of functions is not allowed. If you do not know what that means, for now, interpret this to mean that inside of your functions you can only use variables that are created in that function. For example, the following code fragment would not be allowed, since variable `x` is not a parameter of function `a_times(a)` nor is it a variable created in function `a_times(a)`. It is a global variable created outside of all functions.

```
def a_times(a):
    result=x*a
    return result

x=float(input("Give me a number: "))
print(a_times(10))
```

1 Concentration Game – 100 points

For this assignment you will implement a modified version of Concentration Game. Check out the Wikipedia page on the Concentration Game here: `https://en.wikipedia.org/wiki/Concentration_(game)`

Note that in this assignment (both here and in the code) I used words "deck" and "board" interchangeably, for basically the same thing.

Before reading the description below, watch the video I made on how your game should play: https://youtu.be/yLNJ98Avyrw I also provide you with a file `A3-example-runs.txt` that contains the example runs (those I typed while making the video).

Watching the video will make following the below description much easier. Note that the video does not have a sound as it is not needed. All the requirements implied by the video and example runs in `A3-example-runs.txt` and the provided code in `a3_xxxxxx.py` and the requirements stated in this assignment should be considered mandatory requirements for the assignment.

You may assume that the player will follow your instructions and enter the correct data type but not necessarily the correct data values for that type. For example, if you ask the player for an even integer between 2 and 52 you may assume that indeed they will enter an integer (rather than a real number say), but your program should test if the player entered an integer in the required range and if it is indeed even number. Your program should prompt the player to repeat the entry until correct input is obtained. You should also fully test your program – for example, what does your program do if the player does something silly, like enteres a location that is already discovered, or enters two same locations ... Watch the video to see how your program should behave in these cases.

Also, think of the design of your program. For example, your program should have a function that displays the current board (I provided that fully), but it should also have a function that displays the board with two new positions revealed (I only provided a header and docstrings for that function). These functions should be called from your game playing function (called `play_game`). Designing your program by decomposing it into smaller subproblems (to be implemented as functions) makes programming easer, less prone to errors and makes your code more readable. You will be graded on these aspects of your program too.

The game is usually played with cards lied down on a table. Your game will not use graphics libraries to display the board, but instead it will print the board with the tools that we have seen in class so far, i.e. printing on Python console. A typical card has a suit and a rank on one side. Let's call that side of a card, a *face side*, and the other case a *dull side*. Instead of suit and a rank each of our cards will have a single character on its face side and that character will be an upper case letter of english alphabet or any special character (e.g those you can find on the upper row of your keyboard). A typical card deck has the same drawing on the dull side of the cards. Instead of a drawing our cards will have a character star, `*`, on their dull side.

From now on we can assume we start off with such a deck with some number of cards. For the deck to be suitable for playing Concentration game, each character that appears in the deck needs to appear on an even number of cards. Furthermore, the deck cannot have cards with `*` on their face sides, as we could not distinguish a face side from an a dull side of such cards. Therefore before playing we need to clean up the deck. In particular, we need to remove all the cards that have a `*` on their faces. In addition, for each character that appears on an odd number of cards, we need to remove exactly one card with that character from the deck. After such a clean-up, the resulting deck can be used for playing Concentration game. We call such a deck, the one where each character appears on even number of cards and where no card has `*` on its face side, a *playable deck*. Finally we call a deck *rigorous* if it is playable and each character of the deck appears exactly two times.

Your game must use a (one dimensional) list as the board, as implied by the provided code in `a3_xxxxxx.py`

The game starts off by asking the player

(1) if they wanted a (rigorous) deck, and thus board, automatically generated

(2) or if they want to start with the deck given in a file.

If option (1) is chosen, you need to first prompt the player for the size of the deck. You should only accept even integers, between 2 and 52 for Option 1. As part of this assignment, I provided you with a fully coded function, called `create_board`, that returns a rigorous deck of the given size. Thus option 1 only plays a game with rigorous decks.

If option (2) is chosen, the deck will be read from a file. The player will enter a name of that file when prompted. I provide you with a couple of practice files. (TAs will test your assignment with these and some other files.) To use them, you need to put those files in the same directory/folder as your program `a3_xxxxxx.py`. As part of this assignment, I provided you with a fully coded function, called `read_raw_board`, that reads the file and returns a list containing the deck of cards specified in the file. This deck however is not necessarily playable. Thus you first need to make it playable. For that you will code a function called `clean_up_board`. I provide a header and docstrings for that function. That function should return a playable deck and that is the deck you will play the game with in Option 2. But before playing the game with it, for the fun of it, you would like to know if the resulting deck is not only playable but also rigorous. For that you will need to complete the function called `is_rigorous` for which I provide you with the header and docstrings.

Finally you will notice that `a3_xxxxxx.py` has a function called `play_game(board)`. That is where you should put your game paying code. The function `play_game` takes a playable deck as input parameter (i.e. a deck created by option 1 or option 2). Once the player completes the game, you should also print the number of guesses it took to solve it and how far it is from the optimal (although impossible without luck) len(board)/2 guesses.

Outside of those functions, i.e. in the `main` you will do initial communication with the player, and whatever else is specified in the `a3_xxxxxx.py`.

1.1 Bonus (10 points)

This assignment offers up to 10% bonus. The bonus is available for the maximum of 30-40 students. See below for a clarification and the reason. In order for a student to have a chance to get the bonus the following needs to happen:

1. She needs to correctly solve/code `clean_up_board` and `is_rigorous` functions. She will know that that is the case by waiting for the grading to be completed and seeing that she did not loose any points on these 2 functions; and

2. She needs to solve those two functions in such a way that each of them does asymptotically better than $c \cdot n^2$ operations where n is the length of the input list l and c is some absolute constant. In other words, each function needs to do less than $O(n^2)$ operations. We will see what that means tomorrow in class. If we do not fully cover that topic, you will need to read ahead to learn about it yourself for bonus; and

3. Finally, she needs to come to my office hours between November 6^{th} to 20^{th} in order to explain her faster solution.

Finally, since I cannot examine more than 30-40 students in 3 office hours, the bonus is available to the first 30-40 students that come see me to explain their solution.