Assignment 4: due 8am on Mon, Oct 28th, 2024

Summary of Instructions

Note	Read the instructions carefully and follow them exactly
Assignment Weight	4% of your course grade
Due Date and time	8am on Monday, Oct 28th, 2024
Important	As outlined in the syllabus, late submissions will not be accepted.
	Any files with syntax errors will automatically be excluded from grading. Be sure to test your code before you submit it
	For all functions, both in Part 1 and 2, 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 16-20 of course outline (i.e. slides of Lecture 1). You can find that file on Brightspace under Lecture 1. While at it, also review Course Policy on missing assignments on page 14.

The goal of this assignment is to learn and practice the concepts covered thus far: function design, lists and loops. In this assignment you may not use any of the following:
– dictionaries, sets
– keywords break and continue
– global variables in bodies of functions as explained below.
Using any of these in a solution to a question constitues changing that question. Consequently, that question will not be graded.

Your grade will partially be determined by automatic (unit) tests that will test your functions. All the specified requirements are mandatory (including function names, and behaviour implied by test cases and demo videos). Any requirement that is specified and not met may/will result in deduction of points.

Submit your assignment by the deadline via Brightspace (as instructed and practiced in the first lab.) You can make multiple submissions, but only the last submission before the deadline will be graded. For this assignment you must submit 6 files as described below. For each missing file there will be a grade deduction:

For this assignment, you **do not** need to submit `a4_xxxxxx.txt` as proof that you tested your functions. By now we trust that you learnt and understand the need for and importance of testing your functions and code in general.

The assignment has two parts. Each part explains what needs to be submitted. In part 1 you will implement 4 shorter programs. In part 2, you will implement a card game. Put all the below six required documents (and your declaration file) into a folder, zip it the folder, and submit the resulting `a4_xxxxxx.zip` as explained in lab 1. In particular, the folder (and thus your submission) should have the following files:

Part 1:`a4_Q1_xxxxxx.py, a4_Q2_xxxxxx.py, a4_Q3_xxxxxx.py, a4_Q4_xxxxxx.py`
Part 2: `a4_GAME_xxxxxx.py`
+ `references-YOUR-FULL-NAME.txt`

All programs must run without syntax errors. In particular, when grading your assignment, TAs will first open your file, e.g. `a4_GAME_xxxxxx.py` with IDLE and press Run Module. If pressing Run Module causes any syntax error, the grade for Part 2 becomes zero. The same applies to Part 1.

Furthermore, for each of the functions (in Part 1 and Part 2), I have provided one or more tests to test your functions with. To obtain a partial mark your function may not necessarily give the correct answer on these tests. But if your function gives any kind of python error when run on the given tests, that question will be marked with zero points. Some text cases are given inside docstrings like: `remove_pairs` in the provided starter file `a4_game_xxxxxx.py`; and, `clean_up` and `is_rigorous` in `a4_Q4_xxxxxx.py`.

To determine your grade, your functions will be tested both with examples provided for Part 1 and Part 2 and with some other examples. Thus you too should test your functions with more example than what I provided.

Global variables in bodies of functions are 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, this is not 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))
```

About `references-YOUR-FULL-NAME.txt` file:

The file must be a plain text file. The file must contain references to any code you used that you did not write yourself, including any code you got from a friend, internet, AI engines like chatGPT, social media/forums (including Stack Overflow and discord) or any other source or a person. The only exclusion from that rule is the code that we did in class, the code done as part the lab work, or the code in your textbook. So here is what needs to be written in that file. For every question where you used code from somebody else:

- Write the question number
- . Copy-paste all parts of the code that were written by somebody else. That includes the code you found/were-given and that you then slightly modified.
- Source of the copied code: name of the person or the place on the internet/book where you found it.

While you may not get points for copied parts of the question, you will not be in the position of being accused of plagiarism. Any student caught in plagiarism will receive zero for the whole assignment and will be reported to the dean.

Showing/giving any part of your assignment code to a friend also constitutes plagiarism and the same penalties will apply. If you have nothing to declare/reference, then just write a sentence stating that and put your first and last name under that sentence in your references-YOUR-FULL-NAME.txt file.

Not including references-YOUR-FULL-NAME.txt file, will be taken as you declaring that all the code in the assignment was written by you. Recall though that not submitting that file, comes with a grade penalty.

1 Part 1 (30 points)

For this part of the assignment, you are required to write four short programs. For this part you need to submit 4 files: `a4_Q1_xxxxxx.py`, `a4_Q2_xxxxxx.py`, `a4_Q3_xxxxxx.py` and `a4_Q4_xxxxxx.py` .

1.1 Question 1: (5 points)

Implement a Python function named `number_divisible` that takes a list of integers and a integer `n` as input parameters and returns the number of elements in the list that are divisible by `n`. Then, in the `main`, your program should ask the user to input integers for the list and an integer for `n`, then it should call the function `number_divisible`, and print the result. In this question you may assume that the user will follow your instructions and enter a sequence of integers separated by spaces for the list and an integer for `n`. You can use str method `.strip` and `.split` to handle the user input. Here is a way to ask a user for a list:
`raw_input = input("Please input a list of numbers separated by space: ").strip().split()`
But now `raw_input` is a list of strings that look like integers so you need to create a new list that is a list of equivalent integers.

Function call example:

```
>>> number_divisible([6, 10, 2, 3, 4, 5, 6, 0], 3)
4
```

An example run of the program:

```
Please input a list of integers separated by spaces: 1 2 3 0 5 -6 995
Please input an integer: 2
The number of elements divisible by 2 is 3
```

1.2 Question 2: (5 points)

A run is a sequence of consecutive repeated values. Implement a Python function called `two_length_run` that takes a list of numbers as input parameter and returns `True` if the given list has at least one run (of length at least two), and `False` otherwise. Make sure the function is efficient (i.e. it stops as soon as the answer is known). Then, in the `main`, your program should ask the user to input the list, then it should call `two_length_run` function, and print the result. You can obtain a list of numbers from the user as explained in Question 1.
Four examples of program runs:

```
Please input a list of numbers separated by space: 1     4   3 3  4
True

Please input a list of numbers separated by space: 1 2 3      3   3 4.5 6 5
True

Please input a list of numbers separated by space: 1.0 2 3.7 4 3 2
False

Please input a list of numbers separated by space: 7.7
False
```

Function call examples:

```
>>> two_length_run( [2.7, 1.0, 1.0, 0.5, 3.0, 1.0] )
True
>>>
>>> two_length_run([1.0,1])
True
```

1.3 Question 3: (10 points)

As mentioned, a run is a sequence of consecutive repeated values. Implement a Python function called `longest_run` that takes a list of numbers and returns the length of the longest run. For example in the sequence:
2, 7, 4, 4, 2, 5, 2, 5, 10, 12, 5, 5, 5, 5, 6, 20, 1 the longest run has length 4. Then, in the `main`, your program should ask the user to input the list, then it should call `longest_run function`, and print the result. You can obtain a list of numbers from the user as explained in Question 1.
Five examples of program runs:

```
Please input a list of numbers separated by space: 1 1 2 3.0 3 3 3 3 6 5
5

Please input a list of numbers separated by space:   6 6   7 1 1 1 1 4.5 1
4

Please input a list of numbers separated by space: 6 2.4 4 8 6
1

Please input a list of numbers separated by space: 3
1

Please input a list of numbers separated by space:
0
```

Function call example:

```
longest_run([6, 6, 7, 1.0, 1.0, 1.0, 1, 4.5, 1])
4
```

1.4 Question 4: (10 points)

In the questions you are provided with starter code `a4_Q4_xxxxxx.py` and 5 files `file1.txt, ..., file5.txt`. As usual you cannot modify the given parts of the code. For this question you need to code the two missing functions `clean_up` and `is_rigorous` What these functions should do is described in the docstrings. Some test cases for these two functions can also be found in the docstrings. The provided program asks for the name of a file. The files your program will be tested with will have one character per line (like `bo1.txt`). Then the function `read_raw` returns a list of characters from the file. Here are some example runs:

RUN 1:
```
Enter the name of the file: file1.txt

Before clean-up:
 ['D','F','B','G','$','$','$','A','A','C','G','D','A','$','C','*','P','E','D','*','D','D','E','B','$','#','D','D]

After clean-up:
 ['$', '$', '$', '$', 'A', 'A', 'B', 'B', 'C', 'C', 'D', 'D', 'D', 'D', 'D', 'D', 'E', 'E', 'G', 'G']

This list has no * but is not rigorous and it has 20 characters.
```

RUN 2:
```
Enter the name of the file: file2.txt

Before clean-up:
 ['A', '*', '$', 'C', '*', '*', 'P', 'E', 'D', 'D', '#', 'D', 'E', 'B', '$', '#']

After clean-up:
 ['#', '#', '$', '$', 'D', 'D', 'E', 'E']

This list is now rigorous; it has no * and it has 8 characters.
```

RUN 3:
```
Enter the name of the file: file3.txt

Before clean-up:
 ['A', 'B', '*', 'C', '*', 'D', '*', '*', '*', 'E']

After clean-up:
 []

This list is now rigorous; it has no * and it has 0 characters.
```

RUN 4:
```
Enter the name of the file: file4.txt

Before clean-up:
 ['A', 'A', 'A', 'A', 'A', 'A', 'A']

After clean-up:
 ['A', 'A', 'A', 'A', 'A', 'A']

This list has no * but is not rigorous and it has 6 characters.
```

RUN 5:
```
Enter the name of the file: file5.txt

Before clean-up:
 []

After clean-up:
 []

This list is now rigorous; it has no * and it has 0 characters.
```

2 Part 2: Single Player Rummy Game with Dice and strange deck (80 points)

For this part, you will program a card game described here:
`http://www.classicgamesandpuzzles.com/Old-Maid.html`

You will implement the two player version. One player will be the computer (i.e. your program) and the other a user of your program. In what follows, let's refer to the computer player as Robot and user player as Human. You may assume that Robot will always deal the cards.

As part of this assignment I provided the file called `a4_GAME_xxxxxx.py`. Replace `xxxxxx` in the file name with your student number. You should open that file and run it to see what it does already. All of your code must go inside of that file. The file already has some functions that are fully coded for you and other functions for which only docstrings and partial or no code are provided. 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.

No part of the given code can be changed. Your code must go into clearly indicated places. No code can be added to the main. You can design some extra functions of your own if you like.

Functions `make_deck`, `wait_for_player` and `shuffle_deck` are already fully coded for you.

You need to develop the remaining functions: `deal_cards`, `remove_pairs`, `print_deck`, `get_valid_input`, and `play_game`. The functions must meet the requirements as specified in their docstrings (and as implied by the example program runs below and as implied by the video instructions).

The main bulk of your code (the game playing part) will go into the function called `play_game`. That function should use/call the other functions that you are required to develop (i.e. `deal_cards`, `remove_pairs`, `print_deck`, and `get_valid_input`).

When developing function `get_valid_input` you may assume that Human will enter integer when asked for an integer, but you may not assume that it will be in the correct range.

The function `get_valid_input` gets the input from Human about which face-down card of Robot it wants. When it is Robot's turn to play you must implement it such that Robot takes a random card from Human. Also recall that what Human calls 3rd card, for example, is in position/index 2 in Robot's deck (as it is represented by a list).

Study the example of the program run below carefully to understand how your program should behave. The behaviour of the program that you see in the run is required –s all aspects of it. Also watch the video I made to get even better idea of how your program must behave. The video demo can be found here:

https://youtu.be/mMBApSkvHyM

Some suggestions:

- Study the provided code and understand what it does and how it should be used.
- Spend some time thinking about various parts of the game that need to be implemented. For example, it needs to be able to display Human's deck to Human, it needs to be able to ask Human for what card she wants, it needs to be able to remove pairs from either Human or Robot's deck ... etc. The provided functions do quite of bit of that job for you.
- When you are coding individual functions recall that you can test each function in the shell without finishing the remaining functions. For example, when implementing function `remove_pairs` you can test it in the shell by typing something like:

  ```
  >>> remove_pairs(['10♣', '2♣', '5♦', '6♣', '9♣, 'A♦', '10♦'])
  The shell should display (with cards not necessarily in this order):
  ['2♣', '5♦', '6♣', '9♣', 'A♦']
  ```

  Thus you can code and test your functions one by one (without completing the other parts)
- The game alternates between Robot and Human. Think about how you can represent whose turn it is to play, in your program. One way is to have a variable that you set to zero when it is Robot's turn and to one when it is Human's turn. You also need to figure out what to test to see if the game is over.

2.1 Testing Part 2

Test runs for Part 2 are in the file entitled `A4-game-runs.pdf` The behaviour implied by the tests runs should be considered as required specifications in addition to what is explained above.