Assignment 4: due 8am on Mon, Nov 2, 2020

Summary of Instructions

Note	Read the instructions carefully and follow them exactly
Assignment Weight	7% of your course grade
Due Date and time	8am on Monday, Nov 2, 2020
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 (that do not already have docstrings), 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 15 – 18 of the course outline (i.e. slides of Lecture 1). You can find that file on Brightspace under Lecture 1. While at it, also review Course Policies on pages 13 and 14.

In addition to the assignment files specified later, if you used any code that you did not create/write yourself, your submission will need to contain `declaration-YOUR-FULL-NAME.txt` file. Specifically:

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

It needs to be a plane text file and it must contain references to any code you used that you did not write yourself, including any code you got from a friend, internet, social media/forums (including Stack Overflow and discord) or any other source or person. The only exclusion from that rule is the following: all the material on the Brightspace of the course, anything from your textbook, anything from python.org and python's help pages. So here is what needs to be written in that file. In every question where you used code from somebody else, you must write:
1. question number
2. copy-pasted parts of the code that were written by somebody else. That includes the code you found/were-given that you then slightly modified.
3. whose code it is: name of a person or place on internet/book where you found it.

While you may not get points for that part of the question, you will not be in position of being accused of plagiarism. Not including `declaration-YOUR-FULL-NAME.txt` will be taken as you declaring that all the code in the assignment was written by you. Any student caught in plagiarism will receive zero for the whole assignment and will be reported to the dean. Finally showing/giving any part of your assignment code to a friend also constitute plagiarism and the same penalties will apply.
If you have nothing to declare, you do not need to submit the file `declaration-YOUR-FULL-NAME.txt`

In this assignment you may not use any of the following:
– dictionaries, sets, and
– keywords break and continue
Using any of these in a solution to a question constitues changing that question. Consequently, that question will not be graded.

The goal of this assignment is to learn and practice the concepts covered thus far such as: function design, strings, lists and loops. You can make multiple submissions, but only the last submission before the deadline will be graded. For this assignment, you **do not** need to submit a text file 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. Put all those required documents into a folder called `a4_xxxxxx` where you changed `xxxxxx` to your student number, zip that folder and submit it as explained in Lab 1. In particular, the folder (and thus your submission) should have the following files:

Part 1: `a4_part1_xxxxxx.py`
Part 2: `a4_Q1_xxxxxx.py, a4_Q2_xxxxxx.py, a4_Q3_xxxxxx.py`
+ `declaration-YOUR-FULL-NAME.txt` (if you used code you did not create/write, as detailed above.)

Reminder: Do not submit .rar file instead of .zip file

All programs must run without syntax errors. In particular, when grading your assignment, TAs will first open your file, e.g. `a4_part1_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. You can find tests cases for Part 1 in all of these places: docstrings of the functions in `a4_part1_xxxxxx.py`, then `testRuns_part1.txt` and finally the video.

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. 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))
```

1 Part 1: Mining anagrams and Scrabble help (80 points)

This demo video contains explanations/requirements/instructions on how your program must run:
`https://youtu.be/g5-Fn4cUPi4`

All the program behaviour that you see in the video is required. (Note that youtube allows you to watch at videos at higher or lower speed than normal). Also, find tests cases for Part 1 in all of these places: docstrings of the functions in `a4_part1_xxxxxx.py`, then `testRuns_part1.txt` and the video.

When running your program with the given file `small.txt` and `smallish.txt` it should not take more than 20-30 sec to complete.

For this part of the assignment (part 1) I provided you with file called `a4_part1_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 for patt 1 must go inside of that file in clearly labeled spaces. The file already has some functions that are fully coded for you and other functions for which only docstrings and example runs are provided. No part of the given code can be modified in any way. Only keywords "pass" can be removed. You need to code all of the functions with missing code (i.e. the ones that just have keyword `pass`). You also need to code the `main`, in all the clearly labelled spaces. You can design some extra functions if you like. 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.

For this part of the assignment, you will make a program that figures out anagrams of given words. Knowing anagrams of given words is also helpful when playing scrabble – so your program will also have a helper for scrabble.

Given a sequence of letters, w1, and a sequence of letters, w2, we say that w2 is an anagram of w1 if

1. w2 and w1 have exactly the same letters (i.e. w2 is a permutation of w1) and
2. w2 is a word in english language.

EXAMPLES:
listen is an anagram of silent
knee is an anagram of keen

eekn is NOT an anagram of knee
Even though eekn and knee are comprised of exactly the same letters, they are not anagrams because eekn is not a word in english language.
teen is NOT an anagram of ten because they are not comprised of exactly the same letters

Some hints/suggestions:

I strongly suggest that you code first code ALL the functions in this part before moving to the main. Also I suggest strongly that you do the functions in the given order as earlier functions are needed/helpful in later ones (although you can switch the order in which you do `word_anagrams` and `create_clean_sorted_nodupicates_list`)

`test_letters(w1, w2)` is an important function to code early. The function needs to determined if two words w1 and w2 have exactly the same letters, i.e. that they are permutations of each other. How can you easily determine that? That is, what simple process can you apply to the words and, as a result, know if those words are permutations? Note that the only thing that matters is what letters are in each word, and not their order. Is there an arrangement of the letters that would makes this easy?

It will be useful to convert a string to a list and a list to a string. Here are some examples of that:

```
s='abcd'
q=list(s) # produces list ['a', 'b', 'c', 'd']
q.append('z')
new_str=''.join(q)  # produces string 'abcdz'
```

The string method `join` takes each element from the argument (in this case, `s`) and joins them together using the calling string. If the calling string is the empty string (which it is in this case), it just makes the string. Calling join with a different string, such as `':'` puts that character between each element

```
':'.join(myList) # produces 'a:b:c:d:z'
```

2 Part 2 (20 points)

For this part of the assignment, you are required to write three short programs. For this part you need to submit 3 files: `a4_Q1_xxxxxx.py`, `a4_Q2_xxxxxx.py`, and `a4_Q3_xxxxxx.py`.

2.1 Question 1: (5 points)

Implement a Python function named `number_divisible` that takes a list of integers and a non-zero 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
```

2.2 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 example:

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

2.3 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
```