CSCI 1133, Spring 2023
Homework 05
Due: 11:55 pm, Tuesday February 21, 2023

Due Date: Submit your solutions to Gradescope by 11:55 pm, Tuesday, February 21. Do not upload anything to Canvas and PLEASE be sure to use proper naming conventions for the file, classes, and functions. We will NOT change anything to run it using our scripts.

Late Policy: Submissions will be accepted up to 24 hours late (11:55 pm on Wednesday) for a 1 point penalty, up to 48 hours late (11:55 pm on Thursday) for a 2 point penalty, or up to 72 hours late (11:55 pm on Friday) for a 3 point penalty. Submissions past 11:55 pm on Friday will not be accepted.

Unlike the computer lab exercises, this is not a collaborative assignment. See the syllabus and read section "Academic Dishonesty" for information concerning cheating. Always feel free to ask the instructor or the TAs if you are unsure of something. They will be more than glad to answer any questions that you have. We want you to be successful and learn so give us the chance to help you.

**Purpose:** The primary purpose of this assignment is to get practice with python looping constructs Iterating over both sequences and ranges of numbers.

**Instructions**: This assignment consists of 3 problems, worth a total of 30 points. All documentation and code must be put in a file named hw05.py and submitted on Gradescope.

Because your homework file is submitted and tested electronically, the following are very important:

- Submit the correctly named file through Gradescope by the due date deadline.
- Your program should run without errors when we execute it using Python 3.

Submit your work to Gradescope under the Homework 05 assignment. Make sure that your file is named correctly: if you don't match the file name exactly then the testing script won't work and you'll lose points.

The following will result in a score reduction equal to a percentage of the total possible points:

- Incorrectly named/submitted source file (up to 20%)
- Failure to execute due to syntax errors (up to 30%)
- Bad coding style (missing documentation, meaningless variable names, etc.) (up to 30%)

**A note on break**: While the break keyword can be a useful tool for getting out of loops early, it tends to be misused by intro students to get out of writing a proper loop condition. So for the purposes of this class, the break keyword is considered bad style and will lose you points.

**Documentation:**
Use the following template for EVERY function that you write. The docstring below should be placed inside of the function, just after the function signature.

```
'''
Purpose:
Parameter(s):
Return Value:
'''
```

## Problem A. (10 *points*) **Greater By One**

This problem will exercise your ability with looping constructs over a string.

Write two functions called `greater_by_one_for(a_string)` and `greater_by_one_while(a_string)` Both functions will take in one parameter `a_string`, which represents a string object which contains only digits.

Both functions will also build and **return** a string in which all of the digits between 0 and 8 have been replaced with the digit with a value one greater than the original value. All 9s should be replaced with the digit 0.

**Constraints:**

- In function `greater_by_one_for(a_string)`: You must use a for loop and may not use a while loop
- In function `greater_by_one_while(a_string)`: You must use a while loop and may not use a for loop

**Hints:**

- If we divide an integer n which is less than 10 by 10, we will get a remainder of n.
- If `a_string` is an empty string, then an empty string should be returned.

You are welcome to add an `if __name__ == '__main__'` block and test cases for this and all functions in this assignment but this is not required.

**Examples** (text in **bold** is returned):

```
>>> greater_by_one_for("")
''
>>> greater_by_one_for("123")
'234'
>>> greater_by_one_while("234")
'345'
>>> greater_by_one_for("93023")
'04134'
>>> greater_by_one_while("93023")
'04134'
```

## Problem B. (10 *points*) **Count of Sums**

This problem will exercise your ability with looping constructs over integers.

Write a function called `count_of_sums(lower, upper, sum_val)`. This function will take in three parameters: `lower,` which is an integer that represents the bottom of the searchable range(inclusive), `upper` which is an integer which represents the upper end of the searchable range (inclusive), and `sum_val` which represents the integer sum you are checking against.

This function will **return** an int, representing all of the possible ways to sum two integers between lower and upper and get sum_val. A+B = sum_val and B+A = sum_val count as two different possible ways to sum two integers between lower and upper.

For example, `count_of_sums(2, 10, 8)` would return 5, because there are 5 pairs of integers in the range [2, 10] that sum to 8:

2+6
3+5
4+4
5+3
6+2

**Constraints:**

- You must use a looping construct of some kind to solve this problem.

You are welcome to add an `if __name__ == '__main__'` block and test cases for this and all functions in this assignment but this is not required.

**Examples** (text in **bold** is returned):

```
>>> count_of_sums(2, 10, 8)
5
>>> count_of_sums(0, 15, 15)
16
>>> count_of_sums(-4, 8, 6)
11
>>> count_of_sums(5, 8, 30)
0
```

## Problem C. (10 *points*) **Searching for a string**

This problem will exercise your ability with looping constructs over multiple strings. The goal of this function is given three strings A, B, and C. Find all of the characters that are in both strings A and B, but not in String C.

Write one function called `character_search(string_a, string_b, string_c)` The function will take in three parameters `string_a, string_b, and string_c` all of which will be string objects.

The function will build and **return** a string containing all of the characters that are in `string_a` and `string_b,` but which are not in `string_c`. The output string should have the letters in the same order they appear in `string_a.`

**Hints:**

- You may assume that no character appears in a single string multiple times.

**Constraints**:

- You are not allowed to use the Python set intersection, the `in` keyword to check for inclusion, or anything similar, to trivialize the problem. This should be done only with strings and loops.
    - The `in` keyword when used as required by `for` loop syntax is acceptable

You are welcome to add an `if __name__ == '__main__'` block and test cases for this and all functions in this assignment but this is not required.

**Examples** (text in **bold** is returned):

```
>>> character_search("ABCDEFGHI", "ABCD", "abCd")
'ABD'
>>> character_search("cats dog", "cats", "a")
'cts'
>>> character_search("cats dog", "rats", "a")
'ts'
>>> character_search("cats dog", "cats ", "cats ")
''
```