CS 112 – Fall 2020 – Programming Assignment 11

Recursion

Due Date: Tuesday, November 17th, 11:59 PM

See the **"assignment basics"** file for more detailed information about getting assistance, running the test file, grading, commenting, and many other extremely important things. Each assignment is governed by the rules in that document. Make sure that you are skimming Piazza regularly for any clarifications that get made to the assignment.

Tester File:

- Available in a few days

Background

The purpose of this assignment is to practice recursive functions in Python.

Guidelines

- **Loops are banned.** You **may not** use a for loop or a while loop anywhere in your solution.
- **You must use recursion** in each function in order to receive full credit.
- You are **not** allowed to `import` anything.
- You are **not** allowed to use anything that hasn't been covered in class, including but not limited to: *list comprehension, lambda functions, generators, builtins like zip(), etc.*
- You are **not** allowed to use any string methods.
- Slicing such as lst[1:] **is allowed**.
- The following list methods are **NOT** allowed **list.sort(), list.reverse(), list.insert(), list.remove(), list.index(), list.count(), list.pop(), list.clear(), list.copy().**
- Do not hard code to the examples or to the tester.
- Although you cannot alter the signature of the functions as provided, you can create your **OWN** helper recursive functions as you see fit. **(No Loops!)**
- You should be skimming Piazza regularly in the event that clarifications to this assignment get made. Turn on your email digests!

Testing

In this assignment testing will be done as before. You will start working on the assignment without a tester, and you will do your own testing based on the examples we provide in this document as well as other examples you can come up with based on the provided files. A few days later we will provide an actual tester, but **do not wait** for it in order to start working on the assignment as there won't be enough time to complete it before the deadline. The purpose of the delayed release of the tester is for you to put more emphasis/effort on writing logically correct programs instead of trying to pass certain tests only. When we post the tester, we are going to omit some of the test cases that will be used for grading. **This means that there might be errors in your code even if the tester is giving you no errors.** You must do your own checks to make sure that you haven't missed anything and your code is correct. You do ***not*** need to modify the tester, just test on your own any way you like. Again, the goal is to help you put more focus on writing logically correct programs instead of trying to pass certain tests only.

Grading Rubric

Submitted correctly:	2	# see assignment basics file for file requirements!
Code is well commented:	8	# see assignment basics file for how to comment!
Autograder:	90	# see assignment basics file for how to test!
TOTAL:	100	

Note: If your code does not run and crashes due to errors, it will receive **zero** points. Turning in running code is essential. **Do not submit code with uncommented print statements.**

Assumptions

You may assume that:

- The **types** of the values that are sent to the functions are the proper ones, you do not have to validate them.

Functions

No fancy narrative this time, just a few problems to solve using recursion. **You must use recursion in each function to get full credit.** The use of a loop in one of these functions will result in a zero for that function. *You may not alter the function signatures.*

```
def print_num_pattern(num1, num2)
```

Description: a function that returns a list. The values in the list are created by continually subtracting num2 from num1 until 0 or a negative value is reached, then num2 is continually added to num1, until num1 is back to its original value.

Parameters: `num1` (a positive `int`), `num2` (a positive `int`). `num1` is always greater than `num2`.

Return Value: a list of integers

Examples:

```
print_num_pattern(11, 2) → [11, 9, 7, 5, 3, 1, -1, 1, 3, 5, 7, 9, 11]
print_num_pattern(13, 3) → [13, 10, 7, 4, 1, -2, 1, 4, 7, 10, 13]
```

```
def count_multiples(num1, num2, N)
```

Description: returns the number of multiples of `N` that exist between `num1` and `num2` [inclusive].

Parameters: `num1` (`int`), `num2` (`int`), `N` (`int`). `num1` and `num2` can be in any order. `N` cannot be 0.

Return value: `int`

Examples:

```
count_multiples(2, 13, 3)      → 4  #there are 4 multiples of 3 between 2 to 13
count_multiples(17, -5, 4)     → 6  #there are 6 multiples of 4 between -5 to 17
count_multiples(-10, -5, -3)   → 2  #there are 2 multiples of -3 between -10 to -5
```

```
def scrabble_number(lst)
```

Description: creates a scrabbled version of `lst` by swapping two digits of every consecutive pair of digits within the list.

Parameters: `lst` (a 1 dimensional list of numbers).

Return value: returns a list with scrabbled numbers. [] returned for empty list.

Examples:

```
scrabble_number([-2, -4, 100, 101])  →    [-4, -2, 101, 100]
scrabble_number([1, 2, 3.5, 4.4])    →    [2, 1, 4.4, 3.5]
```

`def count_moves(start, end, step)`

Description: it counts how many moves you need to get from **`start`** to **`end`** with a step of **`step`** by following two rules: 1) if a move lands you on a number ending with 3 or 5, your step is increased by 3 or 5 respectively, 2) if a move lands you on a number ending with 7 or 8, you're automatically transferred ahead by 7 or 8 positions respectively. The last move can land either on **`end`** or any number larger than **`end`**.

Parameters: **`start`** (`int`), **`end`** (`int`), **`step`** (`int`) are all positive. **`end`** is larger or equal to **`start`**.

Return value: `int`

Examples:

```
count_moves(1, 30, 1)      →   5    #1, 2, 3, 7, 18, 37 (5 steps)
count_moves(2, 17, 2)      →   4    #2, 4, 6, 8, 18 (4 steps)
```

`def removeLetter(word, char)`

Description: given a single word and a letter, this function will remove all occurrences of the letter from the word.

Parameters: **`word`** (`string`), **`char`** (`a 1-character string`).

Return value: `string`

Examples:

```
removeLetter('mississippi', 'p') →  mississii
removeLetter('mississippi', 's') →  miiippi
```