CS 112 – Fall 2020 – Programming Assignment 6

Memory and References

Due Date: Sunday, October 11th, 11:59pm

The purpose of this assignment is to practice using loops and working with memory effectively.

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.

Background

Loop statements allow us to run the same block of code repeatedly, with the chance to use different values for variables each time as the accumulated effects pile up. Lists are sequences that can be inspected value-by-value and modified at will. In this assignment we will use loops to perform calculations or adjustments on multidimensional lists.

Restrictions

Any function that violates any of the following will receive **zero** points even if it passes all tests.

- You are **not** allowed to `import` anything
- You are **not** allowed to use slicing
- You are **not** allowed to use `sets` or `dictionaries`
- You are **not** allowed to use any string method
- You are **not** allowed to use anything that hasn't been covered in class
- No built-in function except `range()` and `len()` is allowed
- From list methods, you are allowed to use `.append(), .insert(), .remove()` or `del`. Please **do not ask on piazza** whether you can use `.sort()`, `sorted()`, `.index()`, `.count()` etc.

Testing

From this assignment on, testing will be done a bit different than before. The grader we use to grade your work will be different from the tester we provide. This means that there will be cases that the tester doesn't check and, therefore, your code might have logic errors that the tester doesn't detect. You must do additional checks, on your own, to make sure that you haven't missed anything and your code is correct. You do ***not*** need to modify the tester. The goal is to help you put more focus on writing logically correct programs instead of trying to pass certain tests only. Thus, the grader (a.k.a. autograder) will be made available on Gradescope a few days after the release of this assignment, and will include '**hidden**' test cases.

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.

Functions

The signature of each function is provided below, do ***not*** make any changes to them otherwise the tester will not work properly. The following are the functions you must implement:

`mashup(lst)` **[20pts]**

Description: Creates a new string that is based on the provided list. Each number in the list specifies how many characters from the string that follows belong in the resulting string.

Parameters: **`lst`** is a `list` of variable size, that contains alternating ints and strings

Assumptions: When a pair of values doesn't make sense, throw out that pair. When you have an empty string in the list, move on to the next pair. When you have a number that is larger than the length of the string, move on to the next pair, etc.

Return value: the new `string` that is generated from the replacements

Examples:

```
mashup([2, 'abc', 1, 'def'])                  →   'abd'
mashup([3, 'rate', 2, 'inside', 1, 'goat'])   →   'rating'
```

`expand(numbers, amount)` **[20pts]**

Description: Given a list of **`numbers`** it returns that same list that has been expanded with a certain **`amount`** of zeroes around all of the numbers, including at the beginning and end of the list.

Parameters: **`numbers`** is a `list` of mixed `int` and `float`

Assumptions: You will always have at least one element in numbers. amount will be >= 0

Return value: Nothing is returned. The swapping occurs ***in-place***, i.e. you modify **`numbers`** itself

Examples:

```
ls = [1,2,3]
expand(ls, 1)      # nothing is returned!
print(ls)          # prints [0,1,0,2,0,3,0]

ls = [1.5, -6, 4, 0]
expand(ls, 2)      # nothing is returned!
print(ls)          # prints [0, 0, 1.5, 0, 0, -6, 0, 0, 4, 0, 0, 0, 0, 0]
```

Assumptions for the following two problems:

There will be at least one row and one column in the matrix. All rows will have the same number of elements.

`squarify(matrix)` **[25pts]**

Description: Determine the size of the largest square that can be made with the given matrix. Construct a new square matrix of this size using the elements from the original matrix in their original order.

Parameters: **`matrix`** (`list` of `lists` of `int`)

Return value: A new matrix (`list` of `lists` of `int`)

Examples:

```
ls = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
new_ls = squarify(ls)
print(new_ls)          # prints [[1, 2, 3], [5, 6, 7], [9, 10, 11]]
```

`apply(mask, matrix)` **[25pts]**

Description: Given a **`matrix`**, apply the **`mask`**. The matrix is some MxN list of list of ints, and the mask is exactly a 2x2 list of lists of ints. Imagine you overlay the mask on top of the matrix, starting from the top left corner. There will be 4 places that overlap. Add each pair of numbers that are overlapped, and update the original matrix with this new value. Shift the mask down the row of the matrix to the next 2x2 that hasn't been updated already, and continue this process. Keep doing this down the columns as well. If you are on an edge and only a piece of the mask overlaps, you can ignore the other numbers and only update the overlapping portion.

Parameters: **`matrix`** (MxN `list` of `list` of `ints`) and **`mask`** (2x2 `list` of `list` of `ints`)

Return value: Nothing is returned. The updating occurs ***in-place***, i.e. you modify **`matrix`** itself

Examples:

```
ls = [[1,2],[3,4]]
apply([[1,1],[1,1]], ls)    # nothing is returned!
print(ls)                   # prints [[2, 3], [4, 5]]

ls = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
apply([[1,1],[1,1]], ls)    # nothing is returned!
print(ls)                   # prints [[2, 3, 4], [5, 6, 7], [8, 9, 10]]

ls = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
apply([[1,0],[0,1]], ls)    # nothing is returned!
print(ls)                   # prints [[2, 2, 4], [4, 6, 6], [8, 8, 10], [10, 12, 12]]
```