15-110 Hw3 - Programming Portion

Each of these problems should be solved in the starter file available on the course website. Submit your code to the Gradescope assignment Hw3 - Programming for autograding.

All programming problems may also be checked by running the starter file, which calls the function `testAll()` to run test cases on all programs.

#1 - `addToAll(lst, x)` - 5pts

Write the function `addToAll(lst, x)` which takes a list of numbers and a number and **destructively** modifies the list so that every element has `x` added to it. For example, if `lst = [1, 2, 3]`, calling the function `addToAll(lst, 2)` will change `lst` to hold `[3, 4, 5]`.

#2 - `letterFrequency(s)` - 5pts

Write a function `letterFrequency(s)` which takes a string and returns a list of 26 elements, where each element is the number of times that the corresponding letter of the alphabet occurs in the string. The 0th index corresponds to "a", the 1st corresponds to "b", etc., until the 25th element corresponds to "z".

For example, `letterFrequency("Hello World")` should return: `[0,0,0,1,1,0,0,1,0,0,0,3,0,0,2,0,0,1,0,0,0,0,1,0,0,0]`.

Note: you can ignore any non-letter characters that occur in the string, but you should make sure both upper- and lower-case letters are counted as the same.

Hint: the easiest way to get an index based on a letter is to use the `ord(c)` method, which takes a one-character string as input and returns the ASCII value of that character. Offset this number by `ord("a")` or `ord("A")` to get the index you need.

#3 - `onlyPositive(lst)` - 5pts

Write a function `onlyPositive(lst)` that takes as input a 2D list and returns a new 1D list that contains only the positive elements of the original list, in the order they originally occurred. You may assume the list only has numbers in it.

Example: `onlyPositive([[1, 2, 3], [4, 5, 6]])` returns `[1, 2, 3, 4, 5, 6]`, `onlyPositive([[0, 1, 2], [-2, -1, 0], [10, 9, -9]]` returns `[1, 2, 10, 9]`, and `onlyPositive([[-4, -3], [-2, -1]])` returns `[ ]`.

#4 - `recursiveCount(lst, item)` - 5pts

Write a function `recursiveCount(lst, item)` that takes a list and a value as input and returns a count of the number of times that item occurs in the list. This function must use **recursion** in a meaningful way; a solution that uses a loop or built-in count functions will receive no points.

For example, `recursiveCount([2, 4, 6, 8, 10], 6)` returns `1`,
`recursiveCount([4, 4, 8, 4], 4)` returns `3`, and
`recursiveCount([1, 2, 3, 4], 5)` returns `0`.

#5 - `recursiveMax(lst)` - 5pts

Write a function `recursiveMax(lst)` that takes a list as input and returns the maximum value in the list. You may assume the list contains at least one element. This function must use **recursion** in a meaningful way; a solution that uses a loop or built-in max functions will receive no points.

For example, `recursiveMax([1, 2, 3])` returns `3`, and
`recursiveMax([2, 4, 6, 9, 10, 2, 6])` returns `10`.

Hint: consider what properties the recursive result has if the function works as expected.