CMPSC 132: Programming and Computation II

Lab 3 (10 points) Due date: February 18, 2022, 11:59 PM EST

Goal: The focus of the lab is on recursion, which we discuss in this module. By completing this lab, you should get a better grasp on how to simplify a complex problem towards its base case(s).

General instructions:

- The work in this assignment must be your own original work and be completed alone.
- The instructor and course assistants are available on Teams and with office hours to answer any questions you may have. You may also share testing code on Teams.
- A doctest is provided to ensure basic functionality and may not be representative of the full range of test cases we will be checking. Further testing is your responsibility.
- Debugging code is also your responsibility.
- You may submit more than once before the deadline; only the latest submission will be graded.

Assignment-specific instructions:

- Download the starter code file from Canvas. Do not change the function names or given starter code in your script.
- Each question has different requirements, read them carefully and ask questions if you need clarification. No credit is given for code that does not follow directions.
- You must use recursion to implement these methods. **No credit** will be given if the functions are not recursive, use loops or global variables.
- If you are unable to complete a function, use the pass statement to avoid syntax errors

Submission format:

- Submit your code in a file named LAB3.py file to the Lab 3 Gradescope assignment before the due date.
- As a reminder, code submitted with syntax errors does not receive credit, please run your file before submitting.

get_count(aList, item) (1.5 points)

Returns the number of times item appears in aList. You cannot make any assumptions about the contents of the list. You are not allowed to use any count methods from the Python library.

Input		
list	aList	A list of elements of any type
many	item	An object of any type

Output	
int	Recurrence of item in aList

```
>>> get_count([1,4,3.5,'1',3.5, 9, 1, 4, 2], 1)
2
>>> get_count([1,4,3.5,'1',3.5, 9, 4, 2], 3.5)
2
>>> get_count([1,4,3.5,'1',3.5, 9, 4, 2], 9)
1
>>> get_count([1,4,3.5,'1',3.5, 9, 4, 2], 'a')
0
```

replace(numList, old, new) (1.5 points)

Returns a copy of *numList* with all the occurrences of *old* replaced by *new*. The process should not mutate the original list (alter the memory block referenced by *numList*). You can assume *numList* is a list of numbers and that *old* and *new* are either int or float. You are not allowed to use any replace methods from the Python library.

Input		
list	numList	A list of numbers
int/float	old	A number
int/float	new	A number

Output	
list	Copy of *numList* with all the occurrences of *old* replaced by *new*

```
>>> input_list = [1, 7, 5.6, 3, 2, 4, 1, 9]
>>> replace(input_list, 1, 99.9)
[99.9, 7, 5.6, 3, 2, 4, 99.9, 9]
>>> input_list
[1, 7, 5.6, 3, 2, 4, 1, 9]
>>> replace([1,7, 5.6, 3, 2, 4, 1, 9], 5.6, 777)
[1, 7, 777, 3, 2, 4, 1, 9]
>>> replace([1,7, 5.6, 3, 2, 4, 1, 9], 8, 99)
[1, 7, 5.6, 3, 2, 4, 1, 9]
```

cut(aList) (3 points)

Takes a list of integers and returns a list identical to *aList*, but when a negative number appears, the function deletes the negative number and the next *x-1* elements, where *x* is the absolute value of the negative number. The function should not mutate *aList*.

Input		
list	aList	A list that contains only integers

Output	
list	A list identical to aList where we delete a number of elements from that sequence

```
>>> cut([7, 4, -2, 1, 9])   # Found -2: Delete -2 and 1
[7, 4, 9]
>>> cut([-4, -7, -2, 1, 9]) # Found -4: Delete -4, -7, -2 and 1
[9]
```

neighbor(n) (4 points)

Takes a positive integer and returns an integer where neighboring digits of the same value are removed. Converting *n* to other types such as *str* is not allowed. As described in Homework 1, floor division (//) and modulo (%) operators are useful here. You are not allowed to use the math module.

Input		
int	n	A positive integer

Output	
int	Integer with neighboring duplicates removed

```
>>> neighbor(2222466666678)
24678
>>> neighbor(2222466666625) # Note that the last 2 is not removed
24625
```