Assignment 1: Generating permutations in lexicographic order.

Goal: Write a Python program to generate all the permutations of the characters in a string. This will give you a chance to review some simple Python constructs, i.e.. Strings and Lists and solidify your understanding of recursion.

Your program **must** meet the following specification. You are to write a Python function **perm_gen_lex** that:

- Takes a string as a single input argument. You may assume the string consists of distinct lower case letters (in alphabetical order). You may assume the input is a string of letters in alphabetical order.
- Returns is a **list** of strings where each string represents a permutation of the input string. The list of permutations must be in lexicographic order. (This is basically the ordering that dictionaries use. Order by the first letter (alphabetically), if tie then use the second letter, etc.
- Is well structured, commented, and easy to read. Contains a docstring explaining its purpose
- Is recursive and follows the pseudo code given
- Sample inputs and corresponding outputs:

```
Argument: ‘’         Returns: []
Argument: ‘a’        Returns: [‘a’]
Argument: ‘abc’      Returns: [‘abc’, ‘acb’, ‘bac’, ‘bca’, ‘cab’, ‘cba’]
```

Pseudo code for a recursive algorithm to generate permutations in lexicographic order. **You must follow this pseudo code.**

```
If the string contains a single character return a list containing that string

Loop through all character positions of the string containing the characters to be
permuted, for each character
      Form a simpler string by removing the character
      Generate all permutations of the simpler string recursively
      Add the removed character to the front of each permutation of the simpler word, and
add the resulting permutation to a list

Return all these newly constructed permutations
```

Submit you Python function in a file called **perm_lex.py** and your test program in **perm_lex_testcases.py** to Canvas.

Note: For a string with n characters, you program will return a list contain n! strings. Note that n! grows very quickly. For example : 15! is roughly $1.3*10^{12}$. Thus, it is probably **not** a good idea to test your program with long strings.

Part2: Recursive Lists

Goal: Write a Python program that implements a recursive list structure, along with two recursive functions that operate on the list structure.

The following Node class and StrList data definitions are provided in the starter code:

```
# Node list is
    # None or Node(value, rest), where rest is the rest of the list

class Node:
    def __init__(self, value, rest):
        self.value = value
        self.rest = rest
    # a StrList is one of
    # - None, or
    # - Node(string, StrList)
```

Complete the code for the first_string() function, which will *recursively* search the StrList for the first string in the list. In this case, the first string is determined by comparing the string values using Python's string comparison. For example, the statement "abc" < "ace" evaluates to True in Python, so "abc" would be before "ace". For a list consisting of the strings: "foo", "hello", "Luke" and "805", the first string is "805", as it is "less than" all of the other strings.

```
    # StrList -> string
     # Returns first (as determined by Python compare) string in StrList
    # If StrList is empty (None), return None # Must be implemented recursively
    def first_string(strlist):
```

Complete the code for the split_list() function, which will *recursively* split the StrList into three separate StrLists, returned in a tuple. The first StrList in the returned tuple will contain the strings start with a vowel (a,e,i,o,u), the second will contain the strings that start with a consonant, and the third will contain the strings that don't start with an alpha character. For an input list containing: "Yellow", "abc", "$7.25", "lime", "42", and "Ethan", the result will be:

StrList 1: "abc", "Ethan" StrList 2: "Yellow", "lime" StrList 3: "$7.25", "42"

```
    # StrList -> (StrList, StrList, StrList)
    # Returns a tuple with 3 new StrLists,
    # the first one with strings from the input list that start with a vowel,
    # the second with strings from the input list that start with a consonant,
    # the third with strings that don't start with an alpha character
    # Must be implemented recursively
    def split_list(strlist):
```

Submit your completed rec_list.py and rec_list_testcases.py files to Canvas