CPE 202

Project 1

THERE ARE 3 PARTS TO THIS ASSIGNMENT – MAKE SURE TO DO ALL THREE!

Part1: 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 input string consists 0 or more unique lower-case letters in alphabetical order.
- Returns a Python **list** of strings where each string represents a permutation of the input string. The list of permutations must be in lexicographic order (dictionary order). Note: If you follow the pseudo code below, your list will be constructed such that this condition will be met. **Do not sort the list.**
- Is well structured, commented, and easy to read. Contains a docstring explaining its purpose.
- Is recursive and follows the pseudo code below.

```
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.

```
For each character in the input string:
  Form a simpler string by removing the character from the input string
  Generate all permutations of the simpler string recursively (i.e. call the
    perm_gen_lex function with the simpler string)
  Add the removed character to the front of each permutation of the simpler string, and
    add the resulting permutation to the list
```

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 if you plan on turning the assignment in 'on' time.

Commit and push your completed perm_lex.py and perm_lex_testcases.py files to GitHub

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:

```
# NodeList is
# None or
# Node(value, rest), where rest is the rest of the NodeList
class Node:
    def __init__(self, value: Any, rest: Optional[Node]):
        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: Optional[Node]) -> str:
```

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: Optional[Node]) -> Tuple[Optional[Node],
Optional[Node], Optional[Node]]:
```

Commit and push your completed rec_list.py and rec_list_testcases.py files to GitHub

Part3: A Teddy Bear Picnic

Goal: Determine if winning a game is possible.

This part involves a game with teddy bears. The game starts when I give you some bears. You can then repeatedly give back some bears, but you must follow these rules, where n is the number of bears that you currently have:

1. If n is even, then you may give back n/2 bears.
2. If n is divisible by 3 or 4, then you may multiply the last two digits of n and give back this many bears.
3. If n is divisible by 5, then you may give back 42 bears.

The goal of the game is to end up with EXACTLY 42 bears.

For example, suppose that you start with 250 bears. Then you could make these moves:

- Start with 250 bears.
- Since 250 is divisible by 5, you may return 42 of the bears, leaving you with 208 bears.
- Since 208 is even, you may return half of the bears, leaving you with 104 bears.
- Since 104 is even, you may return half of the bears, leaving you with 52 bears.
- Since 52 is divisible by 4, you may multiply the last two digits (resulting in 10) and return these 10 bears. This leaves you with 42 bears.

You have reached the goal!

Write a recursive function to meet this specification:

```
def bears(n: int) -> bool:
    """A True return value means that it is possible to win
     the bear game by starting with n bears. A False return value means
     that it is not possible to win the bear game by starting with n
     bears."""

Examples:
```

- `bears(250) is True (as shown above)`
- `bears(42) is True`
- `bears(53) is False`
- `bears(41) is False`

Although this problem may seem silly at first, it's an example of a reachability problem, which is a problem that arises in many areas of computer science.https://en.wikipedia.org/wiki/Reachability_problem

Commit and push your completed bears.py and bears_tests.py files to GitHub