Lab 4: Abstract Data Type

1) Stack review

Open *mystack.py* and first review the given stack implementation and the *size* function we discussed in lecture.

Complete the following tasks.

Note that you should write these as top-level functions, not stack methods.

While you may use a temporary stack (as we did in lecture for *size*),do not use any other Python compound data structures, like lists.

1. Write a function that takes a stack of integers and removes all of the items which are greater than 5. The other items in the stack, and their relative order, should remain unchanged.

2. Write a function that takes a stack and returns a new stack that contains each item in the old stack twice in a row. We'll leave it up to you to decide what order to put the copies into in the new stack.

   Note that because the docstring doesn't say that the old stack will be mutated, the old stack should remain unchanged when the function returns.

```
"""CSC148 Lab 4: Abstract Data Types

=== CSC148 Winter 2020 ===
Department of Computer Science,
University of Toronto

=== Module Description ===
In this module, you will write two different functions that operate on
a Stack.
Pay attention to whether or not the stack should be modified.
"""
from typing import Any, List


```

```
14  ##############################################################
15  # Task 1: Practice with stacks
16  ##############################################################
17  class Stack:
18      """A last-in-first-out (LIFO) stack of items.
19
20      Stores data in a last-in, first-out order. When removing an item
    from the
21      stack, the most recently-added item is the one that is removed.
22      """
23      # === Private Attributes ===
24      # _items:
25      #     The items stored in this stack. The end of the list
    represents
26      #     the top of the stack.
27      _items: List
28
29      def __init__(self) -> None:
30          """Initialize a new empty stack."""
31          self._items = []
32
33      def is_empty(self) -> bool:
34          """Return whether this stack contains no items.
35
36          >>> s = Stack()
37          >>> s.is_empty()
38          True
39          >>> s.push('hello')
40          >>> s.is_empty()
41          False
42          """
43          return self._items == []
44
45      def push(self, item: Any) -> None:
46          """Add a new element to the top of this stack."""
47          self._items.append(item)
48
49      def pop(self) -> Any:
50          """Remove and return the element at the top of this stack.
51
52          Raise an EmptyStackError if this stack is empty.
53
54          >>> s = Stack()
55          >>> s.push('hello')
56          >>> s.push('goodbye')
57          >>> s.pop()
58          'goodbye'
59          """
60          if self.is_empty():
61              raise EmptyStackError
62          else:
63              return self._items.pop()
64
65
```

```
    class EmptyStackError(Exception):
        """Exception raised when an error occurs."""
        pass


    def size(s: Stack) -> int:
        """Return the number of items in s.

        >>> s = Stack()
        >>> size(s)
        0
        >>> s.push('hi')
        >>> s.push('more')
        >>> s.push('stuff')
        >>> size(s)
        3
        """
        side_stack = Stack()
        count = 0
        # Pop everything off <s> and onto <side_stack>, counting as we go.
        while not s.is_empty():
            side_stack.push(s.pop())
            count += 1
        # Now pop everything off <side_stack> and back onto <s>.
        while not side_stack.is_empty():
            s.push(side_stack.pop())
        # <s> is restored to its state at the start of the function call.
        # We consider that it was not mutated.
        return count


    # TODO: implement this function!
    def remove_big(s: Stack) -> None:
        """Remove the items in <stack> that are greater than 5.

        Do not change the relative order of the other items.

        >>> s = Stack()
        >>> s.push(1)
        >>> s.push(29)
        >>> s.push(8)
        >>> s.push(4)
        >>> remove_big(s)
        >>> s.pop()
        4
        >>> s.pop()
        1
        >>> s.is_empty()
        True
        """
        pass


    # TODO: implement this function!
```

```
120     def double_stack(s: Stack) -> Stack:
121         """Return a new stack that contains two copies of every item in <
    stack>.
122 
123         We'll leave it up to you to decide what order to put the copies
    into in
124         the new stack.
125 
126         >>> s = Stack()
127         >>> s.push(1)
128         >>> s.push(29)
129         >>> new_stack = double_stack(s)
130         >>> s.pop()  # s should be unchanged.
131         29
132         >>> s.pop()
133         1
134         >>> s.is_empty()
135         True
136         >>> new_items = []
137         >>> new_items.append(new_stack.pop())
138         >>> new_items.append(new_stack.pop())
139         >>> new_items.append(new_stack.pop())
140         >>> new_items.append(new_stack.pop())
141         >>> sorted(new_items)
142         [1, 1, 29, 29]
143         """
144         pass
145 
146 
147     if __name__ == '__main__':
148         import doctest
149         doctest.testmod()
```

Listing 1: mystack.py

2) Queues

3) Running timing experiments

4) Additional exercises