In [5]:

```
# Copyright 2013, Michael H. Goldwasser
#
# Developed for use with the book:
#
#        Data Structures and Algorithms in Python
#        Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser
#        John Wiley & Sons, 2013
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.    See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.    If not, see <http://www.gnu.org/licenses/>.

class LinkedStack:
    """LIFO Stack implementation using a singly linked list for storage."""

    #---------------------------- nested _Node class ----------------------------
    class _Node:
        """Lightweight, nonpublic class for storing a singly linked node."""
        __slots__ = '_element', '_next'                 # streamline memory usage

        def __init__(self, element, next):            # initialize node's fields
            self._element = element                             # reference to user's element
            self._next = next                                   # reference to next node

    #------------------------------- stack methods -------------------------------
    def __init__(self):
        """Create an empty stack."""
        self._head = None                                       # reference to the head node
        self._size = 0                                             # number of stack element

    def __len__(self):
        """Return the number of elements in the stack."""
        return self._size

    def __str__(self):
        obj = self._head
        ret_str = f'{len(self)}: ['
        while obj != None:
            ret_str += str(obj._element)
            obj = obj._next
            if obj != None:
                ret_str += ','

        ret_str += ']'
        return ret_str
    def __repr__(self):
        return self.__str__()

    def is_empty(self):
        """Return True if the stack is empty."""
        return self._size == 0

    def push(self, e):
        """Add element e to the top of the stack."""
        self._head = self._Node(e, self._head)    # create and link a new node
        self._size += 1

    def top(self):
        """Return (but do not remove) the element at the top of the stack.

        Raise Empty exception if the stack is empty.
        """
        if self.is_empty():
```

```
            raise Exception('Stack is empty')
        return self._head._element                          # top of stack is at head of list

    def pop(self):
        """Remove and return the element from the top of the stack (i.e., LIFO).

        Raise Empty exception if the stack is empty.
        """
        if self.is_empty():
            raise Exception('Stack is empty')
        answer = self._head._element
        self._head = self._head._next                       # bypass the former top node
        self._size -= 1
        return answer
```

In [6]:

```
s = LinkedStack()
```

In [7]:

```
s.push('a')
print(s)
s.push('b')
print(s)
s.push('c')
print(s)
```

```
1: [a]
2: [b,a]
3: [c,b,a]
```

In [8]:

```
print(s.pop())
print(s)
print(s.pop())
print(s)
```

```
c
2: [b,a]
b
1: [a]
```

In [ ]:

```

```