In [1]:

```
# 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 LinkedQueue:
    """FIFO queue 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):
            self._element = element
            self._next = next

    #------------------------------ queue methods ------------------------------
    def __init__(self):
        """Create an empty queue."""
        self._head = None
        self._tail = None
        self._size = 0                                                  # number of queue element

    def __len__(self):
        """Return the number of elements in the queue."""
        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 queue is empty."""
        return self._size == 0

    def first(self):
        """Return (but do not remove) the element at the front of the queue.

        Raise Empty exception if the queue is empty.
        """
        if self.is_empty():
            raise Exception('Queue is empty')
        return self._head._element                              # front aligned with head of list
```

```
    def dequeue(self):
        """Remove and return the first element of the queue (i.e., FIFO).

        Raise Empty exception if the queue is empty.
        """
        if self.is_empty():
            raise Exception('Queue is empty')
        answer = self._head._element
        self._head = self._head._next
        self._size -= 1
        if self.is_empty():                                  # special case as queue is empt
            self._tail = None                                  # removed head had been the
tail
        return answer

    def enqueue(self, e):
        """Add an element to the back of queue."""
        newest = self._Node(e, None)                       # node will be new tail node
        if self.is_empty():
            self._head = newest                              # special case: previously empt
        else:
            self._tail._next = newest
        self._tail = newest                                  # update reference to tail node
        self._size += 1
```

In [2]:

```
q = LinkedQueue()
```

In [3]:

```
q.enqueue('a')
print(q)
q.enqueue('b')
print(q)
q.enqueue('c')
print(q)
```

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

In [4]:

```
print(q.dequeue())
print(q)
print(q.dequeue())
print(q)
```

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

In [ ]:

```

```