The Collections Module in Python

Collections is a built-in Python module that implements specialized container datatypes providing alternatives to Python's general purpose built-in containers such as dict, list, set, and tuple. One of the container is dequeue. A deque object support appends and pops from either ends of a list. It is more memory efficient than a normal list object. In a normal list object, the removal of any item causes all items to the right to be shifted towards left by one index. Hence, it is very slow.

Deque objects support the following methods:

1. append(x): Add x to the right side of the deque.
2. appendleft(x): Add x to the left side of the deque
3. clear(): Remove all elements from the deque leaving it with length 0.
4. count(x): Count the number of deque elements equal to x.
5. extend(iterable): Extend the right side of the deque by appending elements from the iterable argument.
6. extendleft(iterable): Extend the left side of the deque by appending elements from iterable. Note, the series of left appends results in reversing the order of elements in the iterable argument.
7. pop(): Remove and return an element from the right side of the deque. If no elements are present, raises an IndexError.
8. popleft(): Remove and return an element from the left side of the deque. If no elements are present, raises an IndexError.
9. remove(value): Remove the first occurrence of value. If not found, raises a ValueError.

```
from collections import deque
d = deque('ghi')

for elem in d:
        print(elem)

d.append('j')
d.appendleft('f')
print(d)
d.pop()
d.popleft()
print(d)
d.extend('jkl')
print(d)
```

Linked List

A linked list is a linear data structure where each element is a separate object. Each element (we will call it a node) of a list is comprising of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the **head/start** of the list. It should be noted that **head** is not a separate node, but the reference to the first node. If the list is empty then the head is a null reference.

Advantage:

A linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand. Any application which has to deal with an unknown number of objects will need to use a linked list.

Disadvantage:

1. One disadvantage of a linked list against an array is that it does not allow direct access to the individual elements. If you want to access a particular item then you have to start at the head and follow the references until you get to that item.
2. Another disadvantage is that a linked list uses more memory compare with an array - we need extra bytes to store a reference to the next node.

```
class Node(object):
  def __init__(self, data=None, next_node=None):
    self.data = data
    self.next_node = next_node

  def get_data(self):
    return self.data

  def get_next(self):
    return self.next_node

  def set_next(self, new_next):
    self.next_node = new_next

class LinkedList(object):
  def __init__(self, head=None):
    self.head = head

            def insertAtBeg(self, data):
                        new_node = Node(data)
                        new_node.set_next(self.head)
                        self.head = new_node

                        def insertAtEnd(self, data):
                        newNode = Node(data)
                        if(self.head):
                                    current = self.head
                                    while(current.next):
                                    current = current.next
                                    current.next = newNode
                        else:
                                    self.head = newNode

            def size(self):
                        current = self.head
                        count = 0
                        while current:
                                    count += 1
                                    current = current.get_next()
                        return count

            #code for counting elements of the list
            def count_elements(self):
                        #create a temporary variable to refer to current node
                        temp = self.head
                        count = 0
                        #iterate until temp is None
                        while temp != None:
                                    count+=1
                                    temp = temp.get_next()
                        return count

            #code for searching elements of the list
            def search_elements(self,data):
                        #create a temporary variable to refer to current node
                        temp = self.head
                        status = False

                        #iterate until temp is None
                        while temp != None:
                                    if temp.get_data() == data:
                                                status = True
                                                break
            temp = temp.get_next()

        return status

    def delete(self, data):
        current = self.head
        previous = None
        found = False
        while current and found is False:
            if current.get_data() == data:
                found = True
            else:
                previous = current
                current = current.get_next()
        if current is None:
            raise ValueError("Data not in list")
        if previous is None:
            self.head = current.get_next()
        else:
            previous.set_next(current.get_next())

lnk_lst = LinkedList()
lnk_lst.insert_at_beg(30)
lnk_lst.insert_at_beg(20)
lnk_lst.insert_at_beg(10)
lnk_lst.show_list()
print("\nTotal elements in list are",lnk_lst.count_elements())
print("10 is","Found" if lnk_lst.search_elements(10) else "Not found")
print("100 is","Found" if lnk_lst.search_elements(100) else "Not found")
```