CS532 Homework 1
Archana Machireddy

Question 1

```
class Element:
    def __init__(self, key):
        self.key = key
        self.next = None
        self.prev = None


class DoublyLinkedList:

    def __init__(self):
        self.head = None

    def insert(self, x):
        x.next = self.head
        if self.head is not None:
            self.head.prev = x
        self.head = x
        x.prev = None

    def delete(self, x):
        if x.prev is not None:
            x.prev.next = x.next
        else:
            self.head = x.next
        if x.next is not None:
            x.next.prev = x.prev

    def search(self, k):
         x = self.head
        while x is not None and x.key != k:
            x = x.next
        if x == None:
            return False
        else:
            return x
```

Question 2

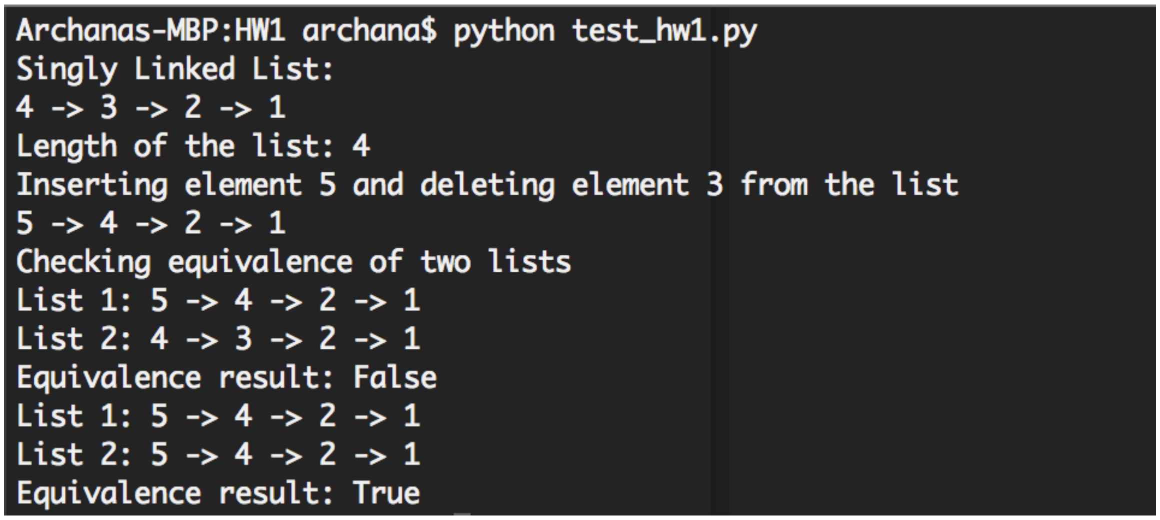
```
class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def insert(self,x):
        # Inserting in the beginning of the list
        x.next = self.head
        self.head = x

    def delete(self, x):
        # If list is empty
        if self.head == None:
            return

        # If head node is the element being searched
        if self.head == x:
            self.head = x.next
            return

        # Search in the rest of the list and find the element
        # before the one to be deleted and change its 'next' link
        data = self.head
        while data is not None:
            if data == x: # First loop will be false (head node)
                prev.next = data.next
                return
            else:
                prev = data
                data = data.next


    def search(self, k):
        x = self.head
        while x is not None and x.key != k:
            x = x.next
        if x == None:
            return False
        else:
            return x
```

The delete operation in singly linked list differs substantially from the one in doubly linked list. In doubly linked list the element being deleted has pointers to both the previous and the next elements making it easier to delete by just modifying these two pointers. In contrast, the singly linked list only has a pointer to its next element. In order to delete an element we need to find its previous element to modify the pointer of the previous element to point to the element next to the element being deleted.

In case only the value of the key to be deleted is given, both the singly and doubly linked lists have to be searched for the key, and the time complexity is O(n) in worst case in both scenarios. But, in case we are given the pointer to the element that has to be deleted, doubly linked list can delete it with time complexity of O(1) as the previous and next pointer are readily available. But in singly linked list even if we have a pointer to the element to be deleted, the list still has to be traversed to find its previous element, making the time complexity O(n).

Question 3

```
def __str__(self):
    x = self.head
    list_contents=[]
    while x is not None:
        list_contents.append(str(x.key))
        x = x.next
    return " -> ".join(list_contents)
```



```
Archanas-MBP:HW1 archana$ python test_hw1.py
Singly Linked List:
4 -> 3 -> 2 -> 1
Length of the list: 4
Inserting element 5 and deleting element 3 from the list
5 -> 4 -> 2 -> 1
Checking equivalence of two lists
List 1: 5 -> 4 -> 2 -> 1
List 2: 4 -> 3 -> 2 -> 1
Equivalence result: False
List 1: 5 -> 4 -> 2 -> 1
List 2: 5 -> 4 -> 2 -> 1
Equivalence result: True
```

Question 4

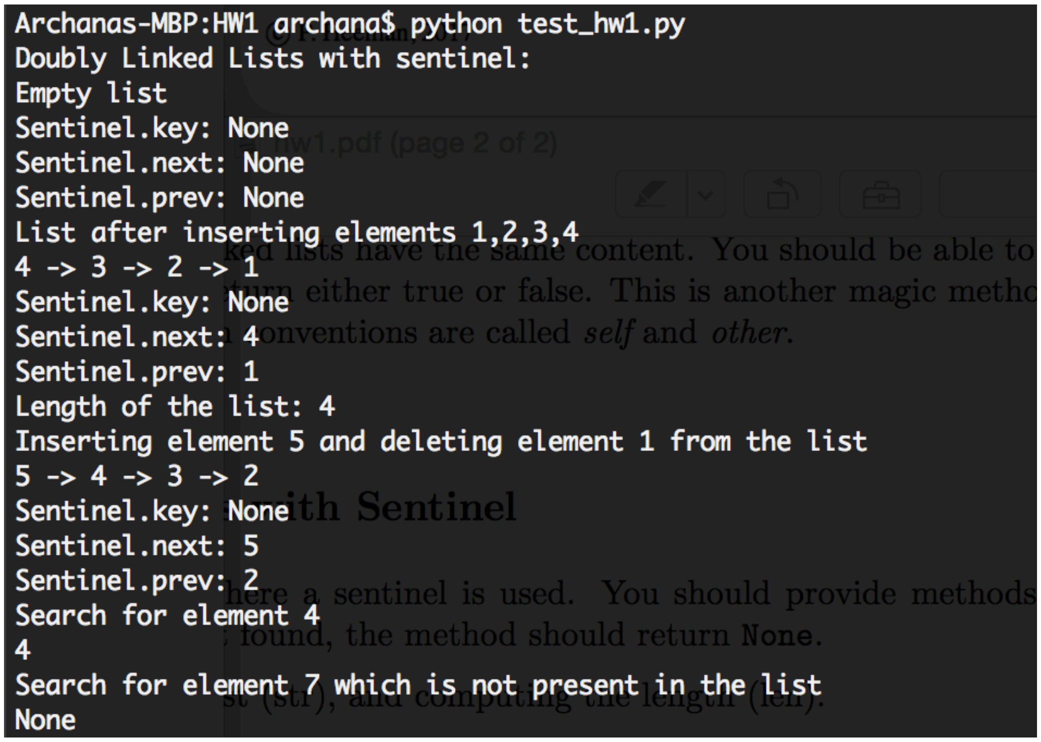
```python
def __len__(self):
    if self.head == None:
        return 0
    count = 0
    x = self.head
    while x is not None:
        count = count+1
        x = x.next
    return count
```

Question 5

```python
def __eq__(self,other):
    x = self.head
    y = other.head
    while x is not None and y is not None:
        if x.key != y.key:
            return False
        else:
            x = x.next
            y = y.next
    if x is None and y is None:
        return True
```

Question 6

```python
class CircularDoublyLinkedList:

    def __init__(self):
        self.sentinel = Element(None)
        self.sentinel.next = self.sentinel
        self.sentinel.prev = self.sentinel

    def insert(self, x):
        x.next = self.sentinel.next
        self.sentinel.next.prev = x
        self.sentinel.next = x
        x.prev = self.sentinel

    def delete(self,x):
        x.prev.next = x.next
        x.next.prev = x.prev
    def search(self, k):
        x = self.sentinel.next
        while x != self.sentinel and x.key != k:
            x = x.next
        if x.key == k:
            return x
        else:
            return None

    def __str__(self):
        x = self.sentinel.next
        list_contents=[]
        while x is not self.sentinel:
            list_contents.append(str(x.key))
            x = x.next
        return " -> ".join(list_contents)

    def __len__(self):
        if self.sentinel.next == self.sentinel:
            return 0
        count = 0
        x = self.sentinel.next
        while x is not self.sentinel:
            count = count+1
            x = x.next
        return count
```



```
Archanas-MBP:HW1 archana$ python test_hw1.py
Doubly Linked Lists with sentinel:
Empty list
Sentinel.key: None
Sentinel.next: None
Sentinel.prev: None
List after inserting elements 1,2,3,4
4 -> 3 -> 2 -> 1
Sentinel.key: None
Sentinel.next: 4
Sentinel.prev: 1
Length of the list: 4
Inserting element 5 and deleting element 1 from the list
5 -> 4 -> 3 -> 2
Sentinel.key: None
Sentinel.next: 5
Sentinel.prev: 2
Search for element 4
4
Search for element 7 which is not present in the list
None
```