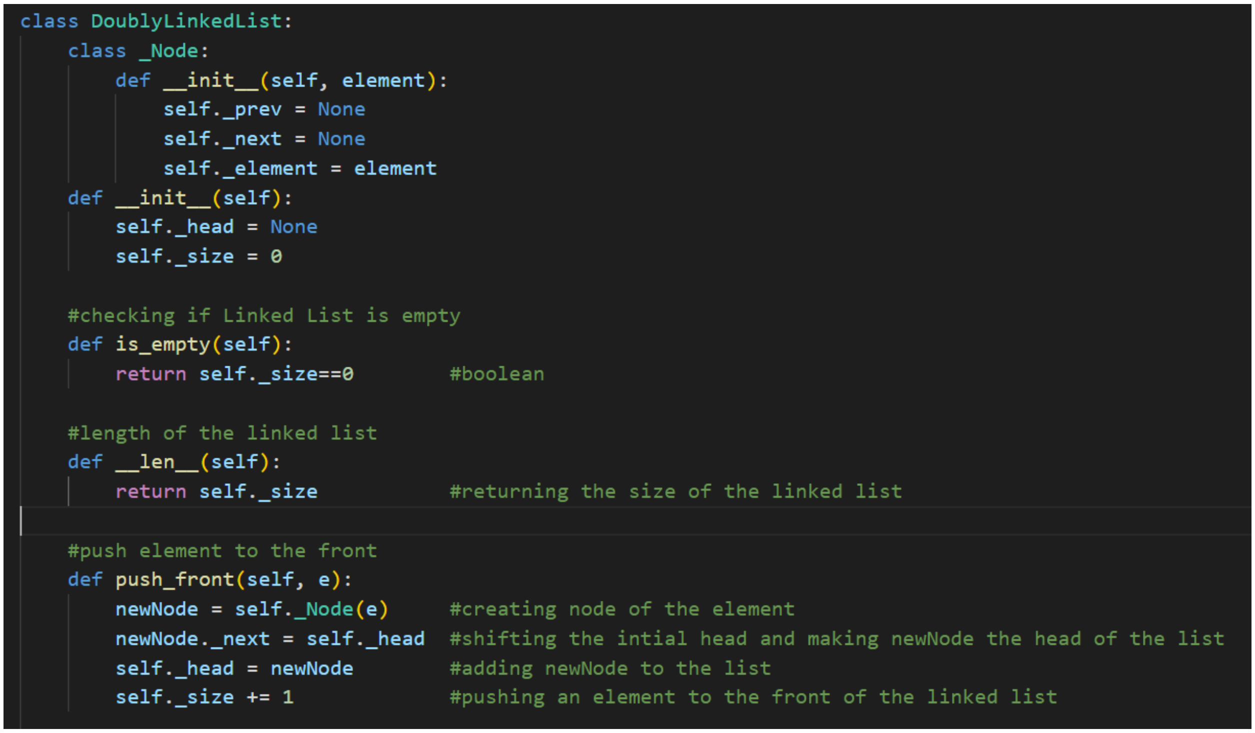
DSA Assignment

By:

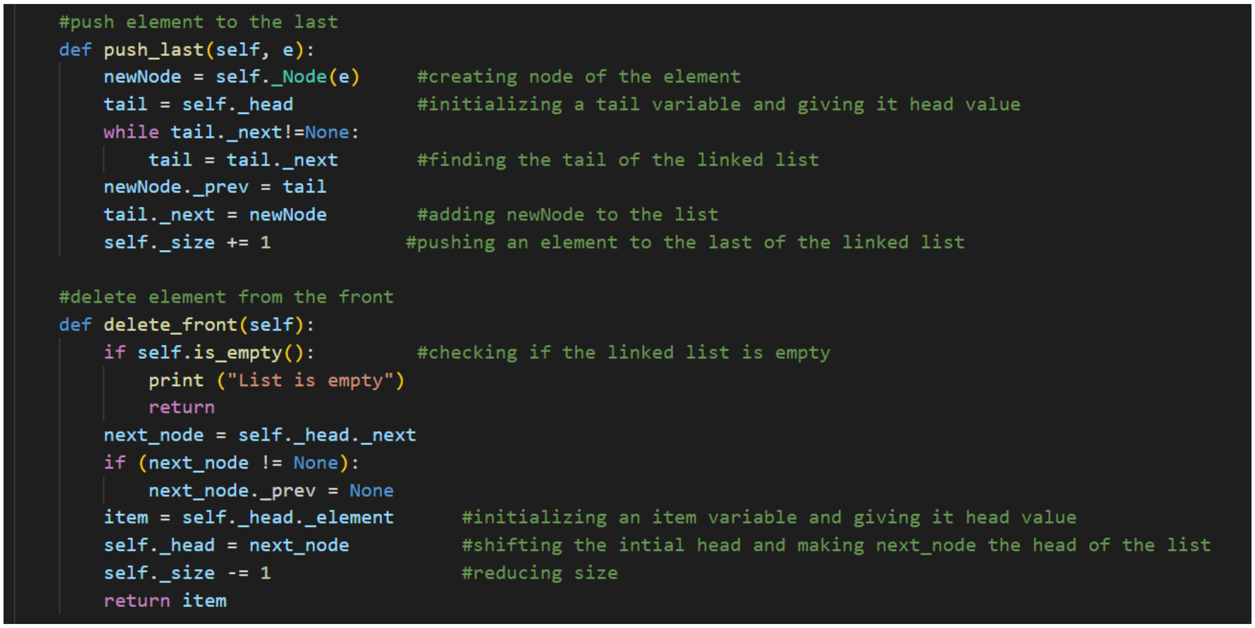
Team Member 1: Luzia Xavier Manuel ; 2021332

Team Member 2: Prashasti Gupta; 2021346

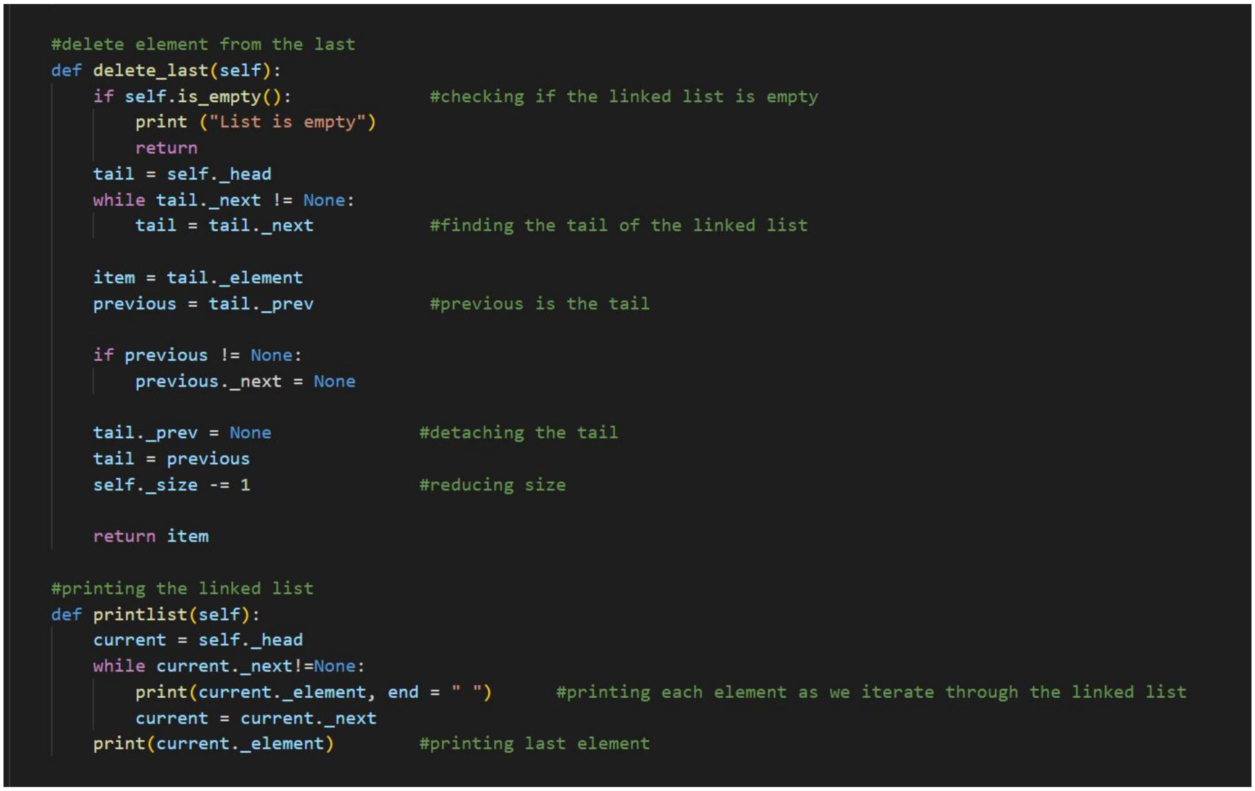
Code:



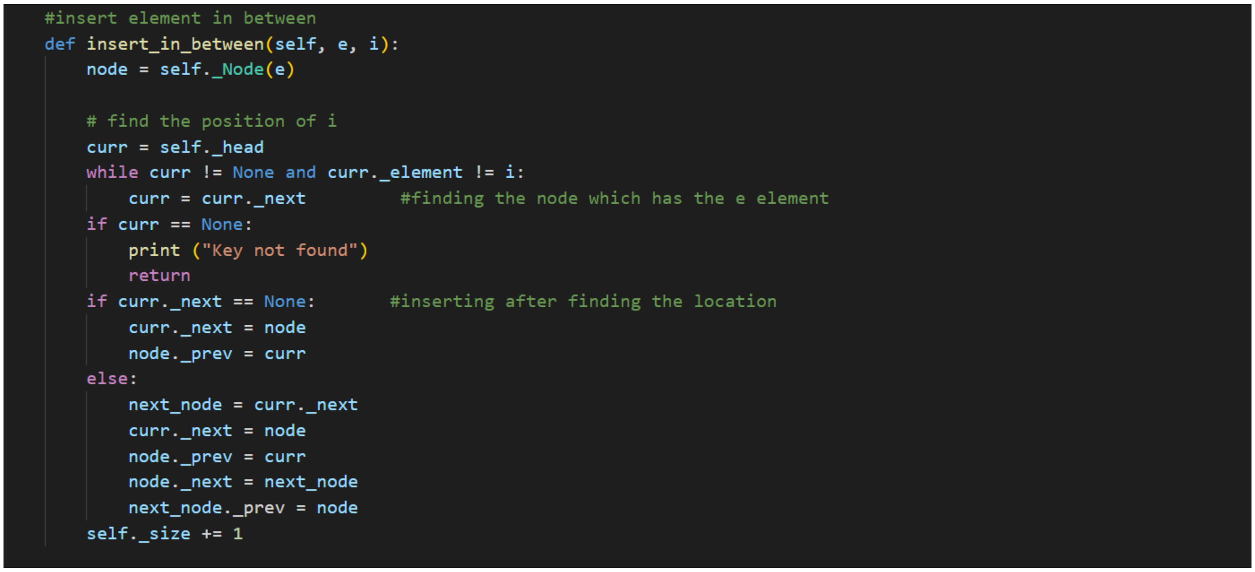
```
class DoublyLinkedList:
    class _Node:
        def __init__(self, element):
            self._prev = None
            self._next = None
            self._element = element
    def __init__(self):
        self._head = None
        self._size = 0

    #checking if Linked List is empty
    def is_empty(self):
        return self._size==0        #boolean

    #length of the linked list
    def __len__(self):
        return self._size           #returning the size of the linked list

    #push element to the front
    def push_front(self, e):
        newNode = self._Node(e)     #creating node of the element
        newNode._next = self._head  #shifting the intial head and making newNode the head of the list
        self._head = newNode        #adding newNode to the list
        self._size += 1             #pushing an element to the front of the linked list
```



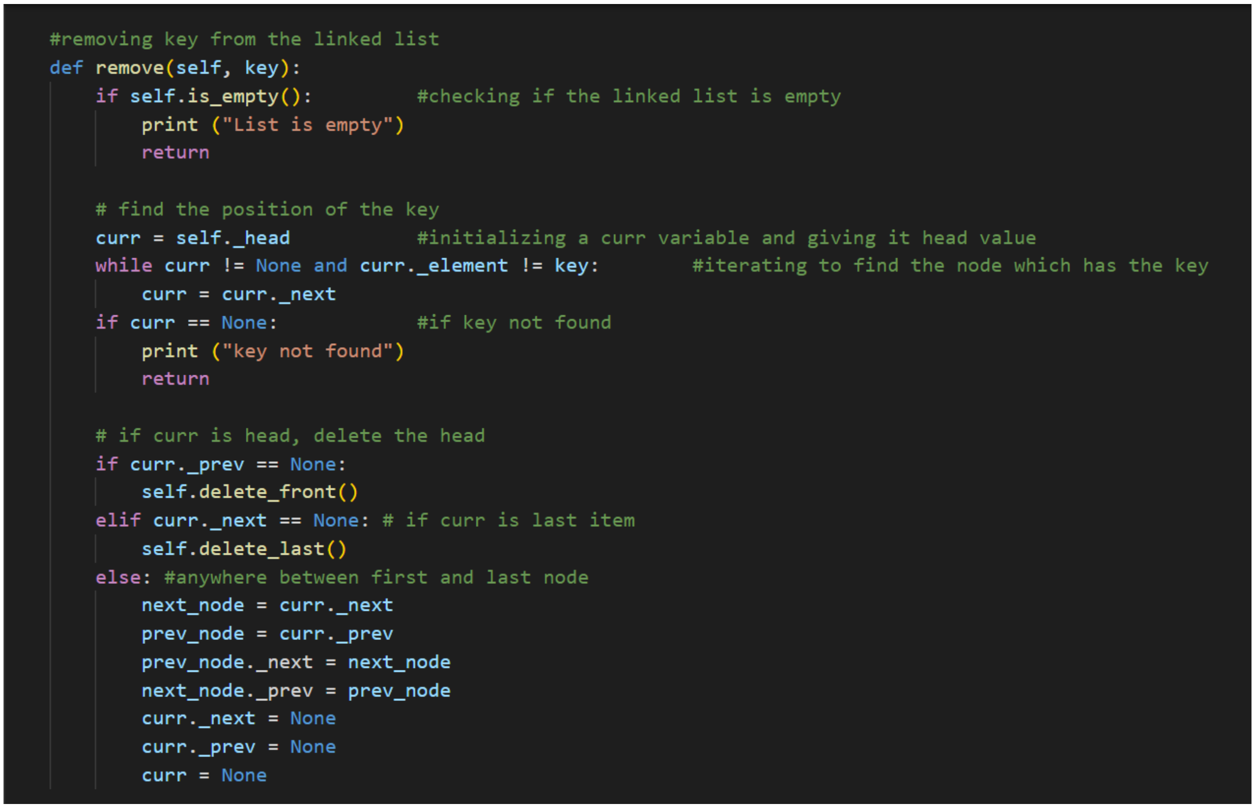
```
    #push element to the last
    def push_last(self, e):
        newNode = self._Node(e)     #creating node of the element
        tail = self._head           #initializing a tail variable and giving it head value
        while tail._next!=None:
            tail = tail._next       #finding the tail of the linked list
        newNode._prev = tail
        tail._next = newNode        #adding newNode to the list
        self._size += 1            #pushing an element to the last of the linked list

    #delete element from the front
    def delete_front(self):
        if self.is_empty():         #checking if the linked list is empty
            print ("List is empty")
            return
        next_node = self._head._next
        if (next_node != None):
            next_node._prev = None
        item = self._head._element      #initializing an item variable and giving it head value
        self._head = next_node          #shifting the intial head and making next_node the head of the list
        self._size -= 1                 #reducing size
        return item
```



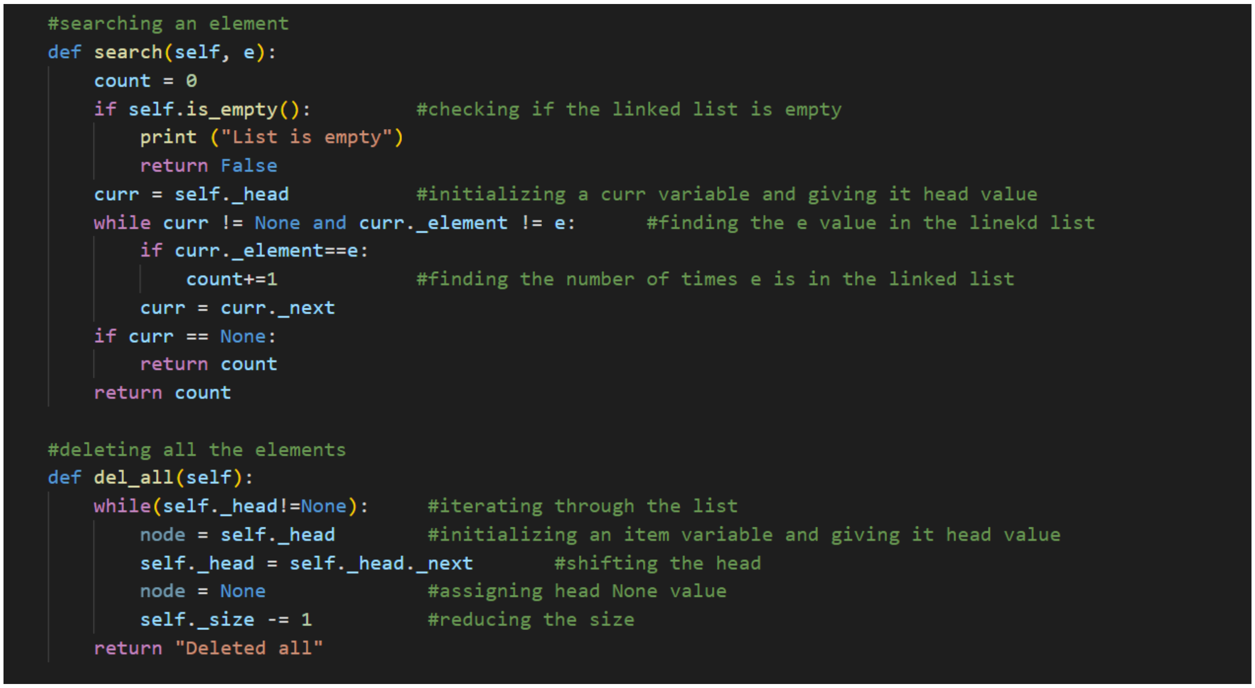
```python
#delete element from the last
def delete_last(self):
    if self.is_empty():             #checking if the linked list is empty
        print ("List is empty")
        return
    tail = self._head
    while tail._next != None:
        tail = tail._next           #finding the tail of the linked list

    item = tail._element
    previous = tail._prev           #previous is the tail

    if previous != None:
        previous._next = None

    tail._prev = None              #detaching the tail
    tail = previous
    self._size -= 1                #reducing size

    return item

#printing the linked list
def printlist(self):
    current = self._head
    while current._next!=None:
        print(current._element, end = " ")      #printing each element as we iterate through the linked list
        current = current._next
    print(current._element)        #printing last element
```



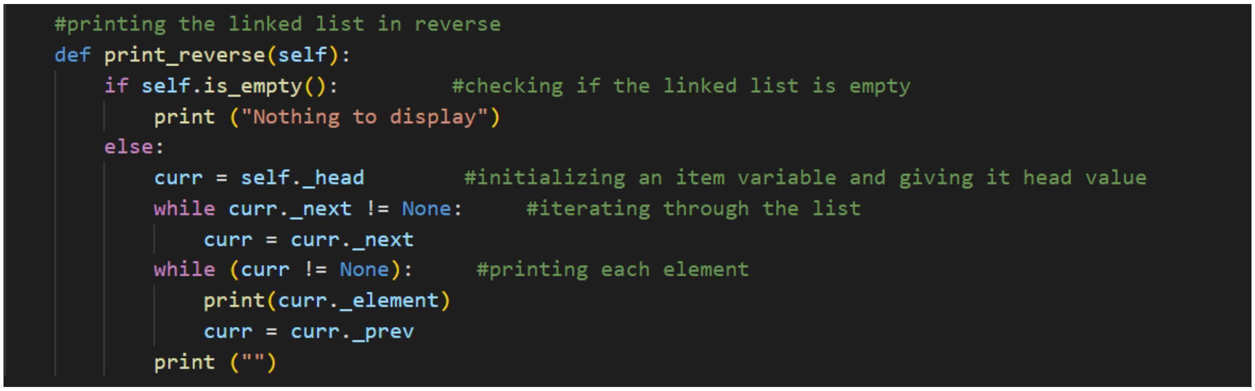
```python
#insert element in between
def insert_in_between(self, e, i):
    node = self._Node(e)

    # find the position of i
    curr = self._head
    while curr != None and curr._element != i:
        curr = curr._next         #finding the node which has the e element
    if curr == None:
        print ("Key not found")
        return
    if curr._next == None:       #inserting after finding the location
        curr._next = node
        node._prev = curr
    else:
        next_node = curr._next
        curr._next = node
        node._prev = curr
        node._next = next_node
        next_node._prev = node
    self._size += 1
```



```
#removing key from the linked list
def remove(self, key):
    if self.is_empty():         #checking if the linked list is empty
        print ("List is empty")
        return

    # find the position of the key
    curr = self._head           #initializing a curr variable and giving it head value
    while curr != None and curr._element != key:        #iterating to find the node which has the key
        curr = curr._next
    if curr == None:            #if key not found
        print ("key not found")
        return

    # if curr is head, delete the head
    if curr._prev == None:
        self.delete_front()
    elif curr._next == None: # if curr is last item
        self.delete_last()
    else: #anywhere between first and last node
        next_node = curr._next
        prev_node = curr._prev
        prev_node._next = next_node
        next_node._prev = prev_node
        curr._next = None
        curr._prev = None
        curr = None
```

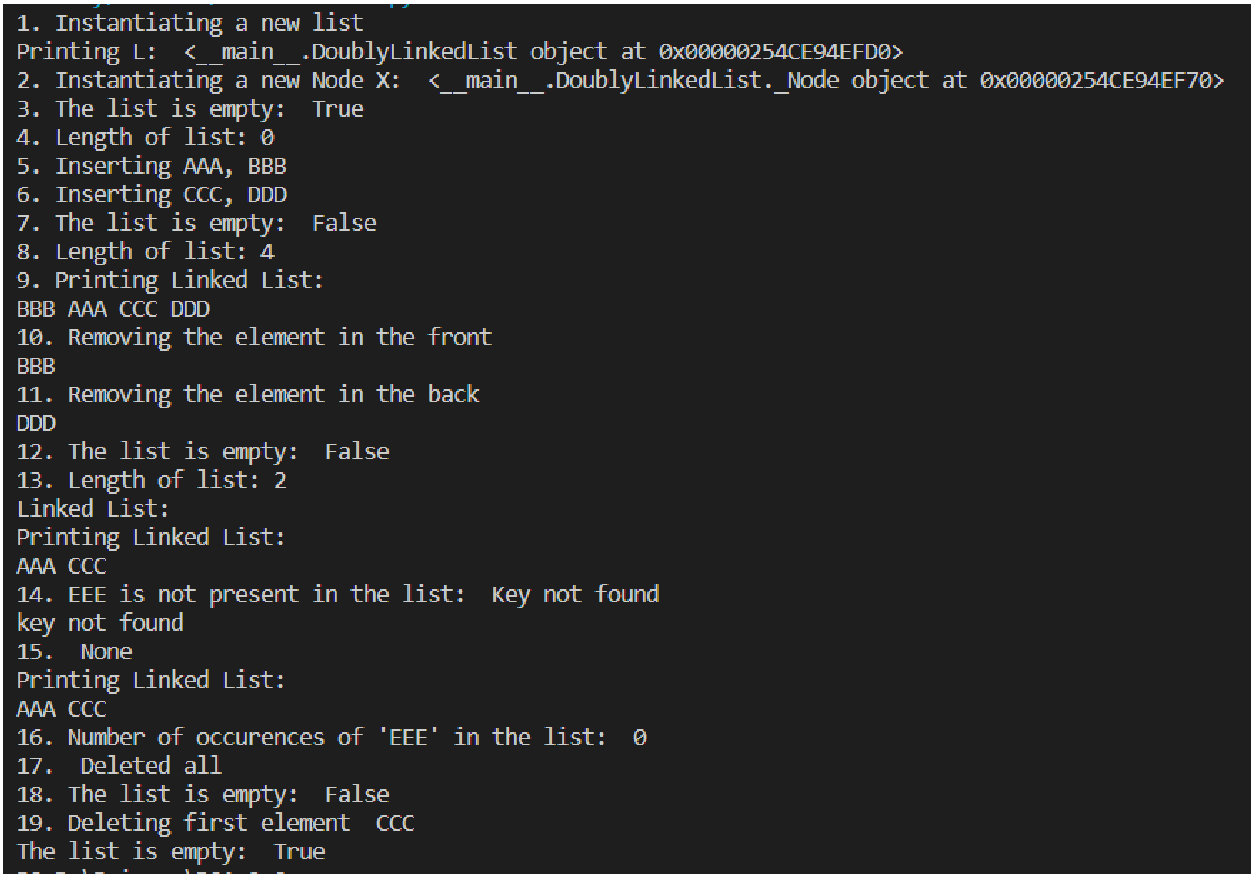


```
#searching an element
def search(self, e):
    count = 0
    if self.is_empty():         #checking if the linked list is empty
        print ("List is empty")
        return False
    curr = self._head           #initializing a curr variable and giving it head value
    while curr != None and curr._element != e:      #finding the e value in the linekd list
        if curr._element==e:
            count+=1            #finding the number of times e is in the linked list
        curr = curr._next
    if curr == None:
        return count
    return count

#deleting all the elements
def del_all(self):
    while(self._head!=None):     #iterating through the list
        node = self._head        #initializing an item variable and giving it head value
        self._head = self._head._next      #shifting the head
        node = None              #assigning head None value
        self._size -= 1          #reducing the size
    return "Deleted all"
```



```
    #printing the linked list in reverse
    def print_reverse(self):
        if self.is_empty():         #checking if the linked list is empty
            print ("Nothing to display")
        else:
            curr = self._head          #initializing an item variable and giving it head value
            while curr._next != None:     #iterating through the list
                curr = curr._next
            while (curr != None):     #printing each element
                print(curr._element)
                curr = curr._prev
            print ("")
```

Output:



```
1. Instantiating a new list
Printing L:  <__main__.DoublyLinkedList object at 0x00000254CE94EFD0>
2. Instantiating a new Node X:  <__main__.DoublyLinkedList._Node object at 0x00000254CE94EF70>
3. The list is empty:  True
4. Length of list: 0
5. Inserting AAA, BBB
6. Inserting CCC, DDD
7. The list is empty:  False
8. Length of list: 4
9. Printing Linked List:
BBB AAA CCC DDD
10. Removing the element in the front
BBB
11. Removing the element in the back
DDD
12. The list is empty:  False
13. Length of list: 2
Linked List:
Printing Linked List:
AAA CCC
14. EEE is not present in the list:  Key not found
key not found
15.  None
Printing Linked List:
AAA CCC
16. Number of occurences of 'EEE' in the list:  0
17.  Deleted all
18. The list is empty:  False
19. Deleting first element  CCC
The list is empty:  True
```