

```
In [2]: 1 class Node:
2         def __init__(self, data=None, next=None):
3             self.data = data
4             self.next = next
5
6
7 class LinkedList:
8     def __init__(self):
9         self.head = None
10
11     def Insert_at_beginning(self, data):
12         node = Node(data, self.head)
13         self.head = node
14
15     def Insert_at_ending(self, data):
16         if self.head is None:
17             self.head = Node(data)
18             return
19         itr = self.head
20         while itr.next:
21             itr = itr.next
22         itr.next = Node(data)
23
24     def insert_values(self, value_list):
25         self.head = None
26         for value in value_list:
27             self.Insert_at_ending(value)
28
29     def get_length(self):
30         count = 0
31         itr = self.head
32         while itr:
33             count += 1
34             itr = itr.next
35         return count
36
37     def insert_at(self, index, data):
38         if index < 0 or index >= self.get_length():
39             raise Exception('invalid index')
40         if index == 0:
41             self.Insert_at_ending(data)
42             return
43         itr = self.head
44         count = 0
45         while itr:
46             if count == index-1:
47                 itr.next = Node(data, itr.next)
48                 break
49             itr = itr.next
50             count += 1
51
52     def remove_at(self, index):
53
54         if index < 0 or index >= self.get_length():
55             raise Exception('invalid index')
56
```

```
57         if index == 0:
58             self.head = self.head.next
59             return
60
61         itr = self.head
62         count = 0
63         while itr:
64             if count == index-1:
65                 itr.next = itr.next.next
66                 break
67             itr = itr.next
68             count += 1
69
70     def insert_after_value(self, data_after, data_to_insert):
71         if self.head is None:
72             return
73         if self.head.data == data_after:
74             self.head.next = Node(data_to_insert, self.head.next)
75             return
76         itr = self.head
77         while itr:
78             if itr.data == data_after:
79                 itr.next = Node(data_to_insert, itr.next)
80                 break
81             itr = itr.next
82
83     def remove_by_value(self, data):
84         if self.head is None:
85             return
86         if self.head.data == data:
87             self.head = self.head.next
88             return
89         itr = self.head
90         while itr.next:
91             if itr.next.data == data:
92                 itr.next = itr.next.next
93                 break
94             itr = itr.next
95
96     def display(self):
97         if self.head is None:
98             print('list is empty')
99             return
100         itr = self.head
101         llstr = ''
102         while itr:
103             llstr += str(itr.data) + '-->'
104             itr = itr.next
105         print(llstr[:-3])
106
107     def search(self, data):
108         itr = self.head
109         while itr.next:
110             if itr.data == data:
111                 return True
112             itr = itr.next
113         return False
```

```
114
115
116 ll = LinkedList()
117 ll.insert_values(["banana", "mango", "grapes", "orange"])
118 ll.display()
119 ll.insert_after_value("mango", "apple")
120 ll.display()
121 ll.remove_by_value("orange")
122 ll.display()
123 ll.remove_by_value("figs")
124 ll.display()
125 ll.remove_by_value("banana")
126 ll.display()
127 print(ll.search("mango"))
128 print("Count: ",ll.get_length())
129 ll.remove_at(1)
130 ll.display()
131 ll.remove_by_value("mango")
132 ll.remove_by_value("apple")
133 ll.remove_by_value("grapes")
134 print("Count: ",ll.get_length())
banana-->mango-->grapes-->orange
banana-->mango-->apple-->grapes-->orange
banana-->mango-->apple-->grapes
banana-->mango-->apple-->grapes
mango-->apple-->grapes
True
Count:  3
mango-->grapes
Count:  0
```

```
In [4]: 1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.next = None
5         self.prev = None
6 class DoublyLinkedList:
7     def __init__(self):
8         self.head = None
9
10    def push(self, new_data):
11        new_node = Node(new_data)
12        new_node.next = self.head
13        if self.head is not None:
14            self.head.prev = new_node
15        self.head = new_node
16
17    def insertAfter(self, prev_node, new_data):
18        if prev_node is None:
19            print ("the given previous node cannot be NULL")
20            return
21        new_node = Node(new_data)
22        new_node.next = prev_node.next
23        prev_node.next = new_node
24        new_node.prev = prev_node
25        if new_node.next is not None:
26            new_node.next.prev = new_node
27
28    def append(self, new_data):
29        new_node = Node(new_data)
30        new_node.next = None
31        if self.head is None:
32            new_node.prev = None
33            self.head = new_node
34            return
35
36        last = self.head
37        while(last.next is not None):
38            last = last.next
39        last.next = new_node
40        new_node.prev = last
41        return
42
43    def printList(self, node):
44        print( "\nTraversal in forward direction")
45        while(node is not None):
46            print (" % d" %(node.data), )
47            last = node
48            node = node.next
49        print( "\nTraversal in reverse direction")
50        while(last is not None):
51            print (" %d " %(last.data), )
52            last = last.prev
53
54    def countNodes(self):
55        counter = 0
56        current = self.head
```

```
57         while(current != None):
58             counter = counter + 1
59             current = current.next
60         return counter
61
62     def search(self, x):
63         current = self.head
64         while current != None:
65             if current.data == x:
66                 return True
67             current = current.next
68         return False
69
70
71 llist = DoublyLinkedList()
72 llist.append(1)
73 llist.push(2)
74 llist.push(3)
75 llist.append(4)
76 llist.insertAfter(llist.head.next, 5)
77 print( "Created DLL is: ", )
78 llist.printList(llist.head)
79 print("\nCount of nodes present in the list: " + str(llist.countNodes()))
80 if llist.search(4):
81     print("Yes")
82 else:
83     print("No")
```

Created DLL is:

Traversal in forward direction

3  
2  
5  
1  
4

Traversal in reverse direction

4  
1  
5  
2  
3

Count of nodes present in the list: 5

Yes

In [ ]: ▶

1