Linked List

Q1. Singly Linked List

```
class Node:
    def __init__(self, dv=None):
        self.data = dv
        self.next = None
    '''def isEmpty(self):
          return'''

    def append(self, dv):
        if self.data == None:
            self.data = dv
            return
        temp = self

        while temp.next != None:

            temp = temp.next

        newnode = Node(dv)
        temp.next = newnode

    # to give the string representation of the object
    def __str__(self):
        items = []
        if self.data == None:
            return str([])
        temp = self
        items.append(temp.data)
```

```
        while temp.next != None:
            temp = temp.next
            items.append(temp.data)
        return str(items)


    def length(self):
        count = 1
        temp = self
        while temp.next != None:
            count = count+1
            temp = temp.next
        return count


    # insertion at the beginning
    def insert_at_beg(self, dv):
        if self.data == None:
            self.data = dv
        else:
            newnode = Node(dv)
            self.data, newnode.data = newnode.data, self.data
            newnode.next = self.next
            self.next = newnode


    # insertion after a node,key-->after value,dv-->insert value
    def insert_after_a_node(self, dv, key):
        if self.data == None:
            return
        temp = self
        while temp.next != None:
```

```
            if temp.data == key:
                newnode = Node(dv)
                newnode.next = temp.next
                temp.next = newnode
            temp = temp.next
            # self.append(dv)
        return


    # insert at any position
    def insert_at_any_position(self, dv, pos):
        if pos == 1:
            self.insert_at_beg(dv)
        elif pos < 1:
            print("position is negative")
            return
        elif pos > self.length():
            print("out of length")
            return
        else:
            count = 1
            temp = self
            # print("count",self.count())
            while count < pos-1:
                temp = temp.next
                count += 1
            newnode = Node(dv)
            newnode.next = temp.next
            temp.next = newnode
        return
```

```
    def insert_before_a_node(self, dv, key):  # insert before a node
        if self.data == None:
            return
        if self.data == key:
            self.insert_at_beg(dv)
            return
        temp = self
        while temp.next.data != key:
            temp = temp.next
            if temp.next == None:
                return
            newnode = Node(dv)
            newnode.next = temp.next
            temp.next = newnode
            return


    def insert_at_last(self, dv):  # insert at the end
        if self.data == None:
            self.data = dv
        temp = self
        while temp.next != None:
            temp = temp.next
        newnode = Node(dv)
        temp.next = newnode
        newnode.next = None
        return
### ----------------------------------------------------------deletion--------------
-------------------------------------------------------------------------###
```

```
def delete_at_the_beg(self):  # delete at the beginning
    if self.data == None:
        return
    if self.next == None:
        self.data = None
        return
    else:
        self.data = self.next.data
        self.next = self.next.next
        return

def delete_at_the_last(self):  # delete at the last
    if self.data == None:
        return
    if self.length() == 1:
        self.data = None
        return
    temp = self
    while temp.next.next != None:
        temp = temp.next
    temp.next = None
    return

def delete_after_a_node(self, key):  # delete after a node
    if self.data == None or self.data == key:
        return
    if self.next == None:
        return
```

```
        temp = self
        while temp.data != key:
            if temp.next == None:
                return
            temp = temp.next
        if temp.next == None:
            return
        temp.next = temp.next.next


    def delete_before_a_node(self, key):  # delete before a node
        if self.data == None or self.data == key:
            return
        if self.next == None:
            return
        if self.next.data == key:
            self.delete_at_the_beg()
        temp = self
        while temp.next.next.data != key:
            if temp.next.next.next == None:
                return
            temp = temp.next
        temp.next = temp.next.next


    def delete_at_any_pos(self, pos):  # delete at any position
        if self.data == None:
            return
        if pos < 1:
            return
        if pos > self.length():
```

```
            return
        if pos == 1:
            self.delete_at_the_beg()
            return
        if pos == self.length():
            self.delete_at_the_last()
            return
        count = 1
        temp = self
        while count < pos-1:
            temp = temp.next
            count += 1
        temp.next = temp.next.next


    def delete_particular_node(self, key):  # delete a particular node
        if self.data == None:
            return
        if self.data == key:
            self.delete_at_the_beg()
            return
        temp = self
        while temp.next.data != key:
            if temp.next.next == None:
                print(key, "not found")
                return
            temp = temp.next
        temp.next = temp.next.next
```

```
### ------------------------------------------------------------creating linked list---
------------------------------------------------------------------------###



sll = Node()
sll.append(10)
print(sll)
sll.append(20)
print(sll)


sll.append(30)
print(sll)
sll.insert_at_beg(50)
print("after insert in beginning:", sll)
sll.insert_at_any_position(200, 3)
print("insert at any position:", sll)
sll.insert_after_a_node(90, 50)
print("insert after a node:", sll)
sll.insert_after_a_node(190, 90)
print("insert after a node:", sll)
sll.insert_before_a_node(180, 190)
print("insert before a node:", sll)
sll.insert_at_last(500)
print("insert at the end of the list:", sll)
sll.delete_at_the_beg()
print("after deleting the 1st node:", sll)
sll.delete_at_the_last()
print("after deleting the last node:", sll)
sll.delete_after_a_node(10)
print("after a specific node:", sll)
```

```
sll.delete_before_a_node(10)
print("delete before a node:", sll)
sll.delete_at_any_pos(2)
print("delete at any position:", sll)
sll.delete_particular_node(1000)
print("delete particular node", sll)
```

Output:

[10]

[10, 20]

[10, 20, 30]

after insert in beginning: [50, 10, 20, 30]

insert at any position: [50, 10, 200, 20, 30]

insert after a node: [50, 90, 10, 200, 20, 30]

insert after a node: [50, 90, 190, 10, 200, 20, 30]

insert before a node: [50, 90, 180, 190, 10, 200, 20, 30]

insert at the end of the list: [50, 90, 180, 190, 10, 200, 20, 30, 500]

after deleting the 1st node: [90, 180, 190, 10, 200, 20, 30, 500]

after deleting the last node: [90, 180, 190, 10, 200, 20, 30]

after a specific node: [90, 180, 190, 10, 20, 30]

delete before a node: [90, 180, 10, 20, 30]

delete at any position: [90, 10, 20, 30]

1000 not found

delete particular node [90, 10, 20, 30]

Q2. Doubly Linked List

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None


class doubly_linked_list:
    def __init__(self):
        self.head = None

# Adding data elements
    def push(self, NewVal):
        NewNode = Node(NewVal)
        NewNode.next = self.head
        if self.head is not None:
            self.head.prev = NewNode
        self.head = NewNode

# Print the Doubly Linked list
```

```
    def listprint(self, node):
        while (node is not None):
            print(node.data),
            last = node
            node = node.next


    def insert(self, prev_node, NewVal):
        if prev_node is None:
            return
        NewNode = Node(NewVal)
        NewNode.next = prev_node.next
        prev_node.next = NewNode
        NewNode.prev = prev_node
        if NewNode.next is not None:
            NewNode.next.prev = NewNode
     def append(self, NewVal):
      NewNode = Node(NewVal)
      NewNode.next = None
      if self.head is None:
         NewNode.prev = None
         self.head = NewNode
         return
      last = self.head
      while (last.next is not None):
         last = last.next
      last.next = NewNode
      NewNode.prev = last
      return
# Define the method to print
```

```
   def listprint(self, node):
      while (node is not None):
         print(node.data),
         last = node
         node = node.next

dllist = doubly_linked_list()
dllist.push(12)
dllist.append(9)
dllist.push(8)
dllist.push(62)
dllist.append(45)
dllist.listprint(dllist.head)

Output:
62 8 12 9 45
```

Q3. Circular Linked List

```
class CNode:
    def __init__(self, dv=None):
        self.data = dv
        self.next = self
    '''def isEmpty(self):
          return'''

    def append(self, dv):
        if self.data == None:
            self.data = dv
```

```
            self.next = self
            return
        temp = self
        while temp.next != self:
            temp = temp.next
        newnode = CNode(dv)
        temp.next = newnode
        newnode.next = self

    # to give the string representation of the object
    def __str__(self):
        items = []
        if self.data == None:
            return str([])
        temp = self
        items.append(temp.data)
        while temp.next != self:
            temp = temp.next
            items.append(temp.data)
        return str(items)

    def length(self):
        count = 1
        temp = self
        while temp.next != self:
            temp = temp.next
            count = count+1
        return count
```

```
    # insertion at the beginning
    def insert_at_beg(self, dv):
        if self.data == self:
            self.data = dv
            self.next = self
        else:
            newnode = CNode(dv)
            self.data, newnode.next = newnode.next, self.data
            # newnode.next=self.next
            # self.data=newnode

    # insertion after a node,key-->after value,dv-->insert value
    def insert_after_a_node(self, dv, key):
        if self.data == None:
            return
        temp = self
        while temp.next != None:
            if temp.data == key:
                newnode = CNode(dv)
                newnode.next = temp.next
                temp.next = newnode
            temp = temp.next
            # self.append(dv)
        return

    # insert at any position
    def insert_at_any_position(self, dv, pos):
        if pos == 1:
            self.insert_at_beg(dv)
```

```
        elif pos < 1:
            print("position is negative")
            return
        elif pos > self.length():
            print("out of length")
            return
        else:
            count = 1
            temp = self
            # print("count",self.count())
            while count < pos-1:
                temp = temp.next
                count += 1
            newnode = CNode(dv)
            newnode.next = temp.next
            temp.next = newnode
        return


    def insert_before_a_node(self, dv, key):  # insert before a node
        if self.data == None:
            return
        if self.data == key:
            self.insert_at_beg(dv)
            return
        temp = self
        while temp.next.data != key:
            temp = temp.next
            if temp.next == None:
                return
```

```
            newnode = CNode(dv)
            newnode.next = temp.next
            temp.next = newnode
            return

    def insert_at_last(self, dv):  # insert at the end
        if self.data == None:
            self.data = dv
        temp = self
        while temp.next != None:
            temp = temp.next
        newnode = CNode(dv)
        temp.next = newnode
        newnode.next = None
        return
### ----------------------------------------------------------deletion--------------
-------------------------------------------------------------------------###

    def delete_at_the_beg(self):  # delete at the beginning
        if self.data == None:
            return
        if self.next == None:
            self.data = None
            return
        else:
            self.data = self.next.data
            self.next = self.next.next
            return
```

```
    def delete_at_the_last(self):  # delete at the last
        if self.data == None:
            return
        if self.length() == 1:
            self.data = None
            return
        temp = self
        while temp.next.next != None:
            temp = temp.next
        temp.next = None
        return


    def delete_after_a_node(self, key):  # delete after a node
        if self.data == None or self.data == key:
            return
        if self.next == None:
            return
        temp = self
        while temp.data != key:
            if temp.next == None:
                return
            temp = temp.next
        if temp.next == None:
            return
        temp.next = temp.next.next


    def delete_before_a_node(self, key):  # delete before a node
        if self.data == None or self.data == key:
            return
```

```
        if self.next == None:
            return
        if self.next.data == key:
            self.delete_at_the_beg()
        temp = self
        while temp.next.next.data != key:
            if temp.next.next.next == None:
                return
            temp = temp.next
        temp.next = temp.next.next


    def delete_at_any_pos(self, pos):  # delete at any position
        if self.data == None:
            return
        if pos < 1:
            return
        if pos > self.length():
            return
        if pos == 1:
            self.delete_at_the_beg()
            return
        if pos == self.length():
            self.delete_at_the_last()
            return
        count = 1
        temp = self
        while count < pos-1:
            temp = temp.next
            count += 1
```

```
        temp.next = temp.next.next


    def delete_particular_node(self, key):  # delete a particular node
        if self.data == None:
            return
        if self.data == key:
            self.delete_at_the_beg()
            return
        temp = self
        while temp.next.data != key:
            if temp.next.next == None:
                print(key, "not found")
                return
            temp = temp.next
        temp.next = temp.next.next


### ------------------------------------------------------------creating linked list---
------------------------------------------------------------------------###


sll = CNode()
sll.append(10)
print(sll)
sll.append(20)
sll.append(30)
sll.append(40)
print(sll)


sll.append(30)
print(sll)
```

```
sll.insert_at_beg(50)
# print("after insert in beginning:", sll)
'''sll.insert_at_any_position(200,3)

print("insert at any position:",sll)
sll.insert_after_a_node(90,50)
print("insert after a node:",sll)
sll.insert_after_a_node(190,90)
print("insert after a node:",sll)
sll.insert_before_a_node(180,190)
print("insert before a node:",sll)
sll.insert_at_last(500)
print("insert at the end of the list:",sll)
sll.delete_at_the_beg()
print("after deleting the 1st node:",sll)
sll.delete_at_the_last()
print("after deleting the last node:",sll)
sll.delete_after_a_node(10)
print("after a specific node:",sll)
sll.delete_before_a_node(10)
print("delete before a node:",sll)
sll.delete_at_any_pos(2)
print("delete at any position:",sll)
sll.delete_particular_node(1000)
print("delete particular node",sll)'''
```

OP:

[10]

[10, 20, 30, 40]

[10, 20, 30, 40, 30]