stack

In [1]:

```

class stack:
    def __init__(self):
        self.item = []
    def display(self):
        return self.item
    def length_item(self):
        return len(self.item)
    def is_empty(self):
        return self.item == []
    def push(self,item):
        self.item.append(item)
    def pop(self):
        return self.item.pop()
    def peek(self):
        return self.item[len(self.item)-1]
    def size(self):
        return len(self.item)
```

In [2]:

```
s = stack()
s.push(10)
s.push(20)
s.push(30)
s.push(40)
s.push(50)
```

In [3]:

```
s.display()
```

Out[3]:

```
[10, 20, 30, 40, 50]
```

In [4]:

```
s.length_item()
```

Out[4]:

```
5
```

In [5]:

```
s.is_empty()
```

Out[5]:

```
False
```

In [6]:

```
s.peek()
```

Out[6]:

```
50
```

In [7]:

```
s.pop()
```

Out[7]:

```
50
```

In [8]:

```
s.display()
```

Out[8]:

```
[10, 20, 30, 40]
```

Queue

In [9]:

```
class Queue:
    def __init__(self):
        self.item = []
    def display(self):
        return self.item
    def peek(self):
        return self.item[-1]
    def is_empty(self):
        return self.item == []
    def enqueue(self,item):
        self.item.insert(0,item)
    def dequeue(self):
        return self.item.pop()
    def size(self):
        return len(self.item)
```

In [10]:

```
q = Queue()
```

In [11]:

```
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
q.enqueue(40)
```

In [12]:

```
q.display()
```

Out[12]:

```
[40, 30, 20, 10]
```

In [13]:

```
q.peek()
```

Out[13]:

```
10
```

In [14]:

```
q.is_empty()
```

Out[14]:

```
False
```

In [15]:

```
q.dequeue()
```

Out[15]:

```
10
```

In [16]:

```
q.size()
```

Out[16]:

```
3
```

Deque

In [17]:

```
#front : last item of a list
#rear : rear is the first index
class Deque:
    def __init__(self):
        self.item = []
    def is_empty(self):
        return self.item == []
    def display(self):
        return self.item
    def peek(self):
        return self.item[-1]
    def add_front(self,item):
        self.item.append(item)
    def add_rear(self,item):
        self.item.insert(0,item)
    def del_front(self):
        self.item.pop()
    def del_rear(self):
        self.item.pop(0)
    def size(self):
        return len(self.item)
```

In [18]:

```
dq = Deque()
```

In [19]:

```
dq.add_front(10)
dq.add_front(20)
dq.add_front(30)
dq.add_front(40)
dq.add_front(50)

dq.add_rear(60)
dq.add_rear(70)
dq.add_rear(80)
dq.add_rear(90)
dq.add_rear(100)
```

In [20]:

```
dq.display()
```

Out[20]:

```
[100, 90, 80, 70, 60, 10, 20, 30, 40, 50]
```

In [21]:

```
dq.del_front()
```

In [22]:

```
dq.del_rear()
```

In [23]:

```
dq.display()
```

Out[23]:

```
[90, 80, 70, 60, 10, 20, 30, 40]
```

In [24]:

```
dq.size()
```

Out[24]:

```
8
```

In [25]:

```
dq.is_empty()
```

Out[25]:

```
False
```

Singly Link list

In [26]:

```
class S_Node:
    def __init__(self, item):
        self.item = item
        self.next = None
```

In [27]:

```
a = S_Node(10)
b = S_Node(20)
c = S_Node(30)
d = S_Node(40)

a.next = b
b.next = c
d.next = d
```

In [28]:

```
class singly_link_list:
    def __init__(self):
        self.head = None
    def print_list(self):
        t = self.head
        while t:
            print(t.item)
            t = t.next
    def pre_append(self,item):
        t = S_Node(item)
        t.next = self.head
        self.head = t
    def post_append(self, item):
        t = S_Node(item)

        if self.head is None:
            self.head = t
            return

        last = self.head
        while last.next:
            last = last.next
        last.next = t
    def append_after_item(self,prev_item, next_item):
        if prev_item is None:
            print("The Link list is empty")
            return

        temp = S_Node(next_item)
        temp.next = prev_item.next
        prev_item.next = temp
    def delete_item(self, value):
        curr = self.head
        if curr and curr.item == value:
            self.head = curr.next
            curr = None
            return

        prev = None
        while curr and curr.item != value:
            prev = curr
            curr = curr.next
        if curr is None:
            return

        prev.next = curr.next
        curr = None
    def delete_item_at_post(self):
        try:
            curr = self.head
            prev = None
            while curr.next != None:
                prev = curr
                curr = curr.next

            prev.next = curr.next
            curr = None
        except:
            print("Single item is not post detetable from Link list or the link list is emp
```

In [29]:

```
s = singly_link_list()
```

In [30]:

```
s.pre_append(10)
s.post_append(40)
s.post_append(50)
s.post_append(60)
s.pre_append(20)
s.pre_append(75)
s.append_after_item(s.head.next,30)
s.append_after_item(s.head.next,35)
s.print_list()
```

```
75
20
35
30
10
40
50
60
```

In [31]:

```
s.delete_item_at_post()
s.print_list()
```

```
75
20
35
30
10
40
50
```

In [32]:

```
s.delete_item(20)
s.print_list()
```

```
75
35
30
10
40
50
```

In [33]:

```
s.delete_item(30)
s.print_list()
```

```
75
35
10
40
50
```

In [34]:

```
s.delete_item_at_post()
s.print_list()
```

```
75
35
10
40
```

In [35]:

```
s.delete_item_at_post()
s.print_list()
```

```
75
35
10
```

In [36]:

```
s.delete_item(75)
s.print_list()
```

```
35
10
```

In [37]:

```
s.delete_item_at_post()
s.print_list()
```

```
35
```

In [38]:

```
s.delete_item_at_post()
s.print_list()
```

```
Single item is not post detetable from Link list or the link list is empty!
35
```

In [39]:

```
s.delete_item(35)
s.print_list()
```

In [40]:

```
s.delete_item_at_post()
s.print_list()
```

```
Single item is not post detetable from Link list or the link list is empty!
```

In [41]:

```
s.post_append(40)
s.post_append(50)
s.post_append(60)
s.print_list()
```

```
40
50
60
```

In [42]:

```
s.pre_append(10)
s.pre_append(20)
s.pre_append(30)
s.print_list()
```

```
30
20
10
40
50
60
```

In [43]:

```
class S_Node:
    def __init__(self,data):
        self.data = data
        self.link = None

class Link_list:
    def __init__(self):
        self.head = None
    def print_list(self):
        t = self.head
        while t:
            print(t.data)
            t = t.link
l = Link_list()
l.head = S_Node(10)
l_2 = S_Node(20)
l_3 = S_Node(30)
l_4 = S_Node(40)
l_5 = S_Node(50)

l.head.link = l_2
l_2.link = l_3
l_3.link = l_4
l_4.link = l_5

l.print_list()
```

```
10
20
30
40
50
```

In [44]:

```
class ListNode(object):
    def __init__(self, x):
        self.val = x
        self.next = None
class LinkedList:
    def __init__(self, head=None):
        self.head = head
    def print_list(self):
        t = self.head
        while t:
            print(t.val)
            t = t.next

def deleteNode(head, node):
    if head == node:
        return None
    ptr = head
    while ptr and ptr.next != node:
        ptr = ptr.next
    if ptr.next == node:
        ptr.next = node.next
    return head
l = LinkedList()
l.head = ListNode(1)
l.head.next = ListNode(2)
l.head.next.next = ListNode(3)
l.head.next.next.next = ListNode(4)
l.print_list()
print("____________")
# This goes from 1->2->3->4 to 1->2->4
head = deleteNode(l.head, l.head.next.next)
l.print_list()
print("____________")
# And this goes to 1->4
#head = deleteNode(head, head)
head = deleteNode(l.head, l.head.next)
l.print_list()
```

```
1
2
3
4
____________
1
2
4
____________
1
4
```

Doubly Link list

In [45]:

```
import gc
```

In [46]:

```
class D_Node:
    def __init__(self, item):
        self.item = item
        self.left = None
        self.right = None
```

In [47]:

```
a = D_Node(10)
b = D_Node(20)
c = D_Node(30)
d = D_Node(40)

a.right = b
b.left = a
b.right = c
c.left = b
c.right = d
d.left = c
```

In [48]:

```
class D_Linklist:
    def __init__(self):
        self.head = None
    def print_list(self):
        t = self.head
        while t:
            print(t.item)
            t = t.right
    def pre_append(self,item):
        new_node = D_Node(item)
        new_node.right = self.head
        new_node.left = None

        if self.head is not None:
            self.head.left = new_node
        self.head = new_node
    def insert_after(self, prev_node, new_data):
        if prev_node is None:
            print("This node does not exist in Doublly link list!")
            return
        temp = D_Node(new_data)
        temp.right = prev_node.right
        prev_node.right = temp
        temp.left = prev_node

        if temp.right is not None:
            temp.right.left = temp

    def post_append(self,item):
        last = self.head
        new_node = D_Node(item)
        new_node.right = None
        if self.head is None:
            new_node.left = None
            self.head = new_node
            return
        while last.right is not None:
            last = last.right
        last.right = new_node

        new_node.left = last
    def delete_node(self, dele):

        if self.head is None or dele is None:
            return

        if self.head == dele:
            self.head = dele.right

        if dele.right is not None:
            dele.right.left = dele.left

        if dele.left is not None:
            dele.left.right = dele.right
        gc.collect()
```

In [49]:

```
D_l = D_Linklist()
```

In [50]:

```
D_l.pre_append(10)
```

In [51]:

```
D_l.pre_append(20)
```

In [52]:

```
D_l.pre_append(30)
D_l.pre_append(40)
D_l.pre_append(50)
```

In [53]:

```
D_l.print_list()
```

```
50
40
30
20
10
```

In [54]:

```
D_l.post_append(60)
```

In [55]:

```
D_l.print_list()
```

```
50
40
30
20
10
60
```

In [56]:

```
D_l.insert_after(D_l.head.right.right, 35)
```

In [57]:

```
D_l.insert_after(D_l.head.right, 25)
```

In [58]:

```
D_l.insert_after(D_l.head.right.left, 15)
```

In [59]:

```
D_l.print_list()
```

```
50
15
40
25
30
35
20
10
60
```

In [60]:

```
D_l.delete_node(D_l.head.right.right.left)
D_l.print_list()
```

```
50
40
25
30
35
20
10
60
```

In [61]:

```
D_l.delete_node(D_l.head.right.right.left.left)
D_l.print_list()
```

```
40
25
30
35
20
10
60
```

In [62]:

```
D_l.delete_node(D_l.head)
D_l.print_list()
```

```
25
30
35
20
10
60
```

In [63]:

```
D_l.delete_node(D_l.head.right)
D_l.print_list()
```

```
25
35
20
10
60
```

In [ ]:

```
```

In [ ]:

```
```