

stack

In [69]:

```
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 [70]:

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

In [71]:

```
s.display()
```

Out[71]:

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

In [72]:

```
s.length_item()
```

Out[72]:

```
5
```

In [73]:

```
s.is_empty()
```

Out[73]:

```
False
```

In [74]:

```
s.peek()
```

Out[74]:

50

In [75]:

```
s.pop()
```

Out[75]:

50

In [76]:

```
s.display()
```

Out[76]:

[10, 20, 30, 40]

Queue

In [77]:

```
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 [78]:

```
q = Queue()
```

In [79]:

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

In [80]:

```
q.display()
```

Out[80]:

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

In [81]:

```
q.peak()
```

Out[81]:

```
10
```

In [82]:

```
q.is_empty()
```

Out[82]:

```
False
```

In [83]:

```
q.dequeue()
```

Out[83]:

```
10
```

In [84]:

```
q.size()
```

Out[84]:

```
3
```

Deque

In [85]:

```
#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 [86]:

```
dq = Deque()
```

In [87]:

```
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 [88]:

```
dq.display()
```

Out[88]:

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

In [89]:

```
dq.del_front()
```

In [90]:

```
dq.del_rear()
```

In [91]:

```
dq.display()
```

Out[91]:

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

In [92]:

```
dq.size()
```

Out[92]:

```
8
```

In [93]:

```
dq.is_empty()
```

Out[93]:

```
False
```

Singly Link list

In [94]:

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

In [95]:

```
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 [96]:

```

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 [97]:

```
s = singly_link_list()
```

In [98]:

```
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 [99]:

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

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

In [100]:

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

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

In [101]:

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

```
75
35
10
40
50
```

In [102]:

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

```
75
35
10
40
```

In [103]:

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

```
75
35
10
```

In [104]:

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

```
35
10
```

In [105]:

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

```
35
```

In [106]:

```
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 [107]:

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


In [108]:

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

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

In [109]:

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

```
40  
50  
60
```

In [110]:

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

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

In [111]:

```
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 [112]:

```

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 [113]:

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

In [114]:

```
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 []:

In []:

In []: