Stack

In [5]:

```
#implementation using list
stack=[]
stack.append("Welcome")
stack.append("to")
stack.append("Data structure")
print(stack)
print(stack.pop())
print(stack)
print(stack.pop())
print(stack)
print(stack.pop())
print(stack)
```

```
['Welcome', 'to', 'Data structure']
Data structure
['Welcome', 'to']
to
['Welcome']
Welcome
[]
```

In [7]:

```
#implementation of stack using dequeu
from collections import deque
stack=deque()
stack.append("Welcome")
stack.append("to")
stack.append("Data structure")
print(stack)
print(stack.pop())
print(stack)
print(stack.pop())
print(stack)
print(stack.pop())
print(stack)
```

```
deque(['Welcome', 'to', 'Data structure'])
Data structure
deque(['Welcome', 'to'])
to
deque(['Welcome'])
Welcome
deque([])
```

In [14]:

```
#implementation of stack using Queue
from queue import LifoQueue
stack=LifoQueue(maxsize=3)
stack.put('4')
stack.put('9')
stack.put('5')
print(stack.qsize())
print(stack.full())
stack.get()
print(stack.full())
```

```
3
True
False
```

Queue

In [2]:

```
#Queue
class Queue:
    def __init__(self):
        self.queue = []
    def enqueue(self,item):
        self.queue.append(item)
    def dequeue(self):
        if len(self.queue) < 1:
            return None
        return self.queue.pop(0)
    def display(self):
        print(self.queue)
    def size(self):
        return len(self.queue)
Q=Queue()
Q.enqueue(1)
Q.enqueue(2)
Q.enqueue(3)
Q.enqueue(4)
Q.enqueue(5)
Q.display()
Q.dequeue()
print("after removing of element ")
Q.display()
Q.dequeue()
print("after removing of element ")
Q.display()
```

```
[1, 2, 3, 4, 5]
after removing of element
[2, 3, 4, 5]
after removing of element
[3, 4, 5]
```

Singly Linked list

In [6]:

```
#Creating a node
class node:
    def __init__(self,data):
        self.data=data
        self.next=None
n1=node(5)
print(n1.data)
print(n1.next)
```

```
5
None
```

In [9]:

```
#creating singly linked list
class node:
    def __init__(self,data):
        self.data=data
        self.next=None

class singlyLL:
    def __init__(self):
        self.head=None
        print("SinglyLL")
sll=singlyLL()
```

```
SinglyLL
```

In [16]:

```
#creating traversal
class node:
    def __init__(self,data):
        self.data=data
        self.next=None

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

    def traversal(self):
        if self.head is None:
            print("Singly LL is empty")
        else:
            a=self.head
            while a is not None:
                print(a.data, end=" ")
                a=a.next
n1=node(7)
sll=SLL()
sll.head=n1
n2=node(6)
n1.next=n2
n3=node(5)
n2.next=n3
n4=node(4)
n3.next=n4
n5=node(3)
n4.next=n5
n6=node(2)
n5.next=n6
n7=node(1)
n6.next=n7
sll.traversal()
```

```
7 6 5 4 3 2 1
```

In [1]:

```
#insertion at beganing in singly linked list
class node:
    def __init__(self,data):
        self.data=data
        self.next=None

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

    def traversal(self):
        if self.head is None:
            print("Singly LL is empty")
        else:
            a=self.head
            while a is not None:
                print(a.data, end=" ")
                a=a.next

    def insert_at_beganing(self,data):
        print()
        nb=node(data)
        nb.next=self.head
        self.head=nb


n1=node(7)
sll=SLL()
sll.head=n1
n2=node(6)
n1.next=n2
n3=node(5)
n2.next=n3
n4=node(4)
n3.next=n4
n5=node(3)
n4.next=n5
n6=node(2)
n5.next=n6
n7=node(1)
n6.next=n7
sll.traversal()
sll.insert_at_beganing(100)
sll.traversal()
```

```
7 6 5 4 3 2 1
100 7 6 5 4 3 2 1
```

In [3]:

```
#insertion at end in singly linked list
class node:
    def __init__(self,data):
        self.data=data
        self.next=None

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

    def traversal(self):
        if self.head is None:
            print("Singly LL is empty")
        else:
            a=self.head
            while a is not None:
                print(a.data, end=" ")
                a=a.next

    def insert_at_beganing(self,data):
        print()
        nb=node(data)
        nb.next=self.head
        self.head=nb
    def insert_at_end(self,data):
        print()
        ne=node(data)
        a=self.head
        while a.next is not None:
            a=a.next
        a.next=ne

n1=node(7)
sll=SLL()
sll.head=n1
n2=node(6)
n1.next=n2
n3=node(5)
n2.next=n3
n4=node(4)
n3.next=n4
n5=node(3)
n4.next=n5
n6=node(2)
n5.next=n6
n7=node(1)
n6.next=n7
sll.traversal()
sll.insert_at_beganing(100)
sll.traversal()
sll.insert_at_end(500)
sll.traversal()
```

```
7 6 5 4 3 2 1
100 7 6 5 4 3 2 1
100 7 6 5 4 3 2 1 500
```

In [7]:

```
#insertion at specified node in singly linked list
class node:
    def __init__(self,data):
        self.data=data
        self.next=None

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

    def traversal(self):
        if self.head is None:
            print("Singly LL is empty")
        else:
            a=self.head
            while a is not None:
                print(a.data, end=" ")
                a=a.next

    def insert_at_beganing(self,data):
        print()
        nb=node(data)
        nb.next=self.head
        self.head=nb

    def insert_at_end(self,data):
        print()
        ne=node(data)
        a=self.head
        while a.next is not None:
            a=a.next
        a.next=ne

    def insert_at_specific_node(self,position,data):
        print()
        nib=node(data)
        a=self.head
        for i in range(1,position-1):
            a=a.next
        nib.next=a.next
        a.next=nib

n1=node(7)
sll=SLL()
sll.head=n1
n2=node(6)
n1.next=n2
n3=node(5)
n2.next=n3
n4=node(4)
n3.next=n4
n5=node(3)
n4.next=n5
n6=node(2)
n5.next=n6
n7=node(1)
n6.next=n7
sll.traversal()
sll.insert_at_beganing(100)
sll.traversal()
sll.insert_at_end(500)
sll.traversal()
sll.insert_at_specific_node(4,75)
sll.traversal()
```

```
7 6 5 4 3 2 1
100 7 6 5 4 3 2 1
100 7 6 5 4 3 2 1 500
100 7 6 75 5 4 3 2 1 500
```

In [5]:

```
#deletion at beganing in singly linked list
class node:
    def __init__(self,data):
        self.data=data
        self.next=None

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

    def traversal(self):
        if self.head is None:
            print("Singly LL is empty")
        else:
            a=self.head
            while a is not None:
                print(a.data, end=" ")
                a=a.next

    def insert_at_beganing(self,data):
        print()
        nb=node(data)
        nb.next=self.head
        self.head=nb

    def insert_at_end(self,data):
        print()
        ne=node(data)
        a=self.head
        while a.next is not None:
            a=a.next
        a.next=ne

    def insert_at_specific_node(self,position,data):
        print()
        nib=node(data)
        a=self.head
        for i in range(1,position-1):
            a=a.next
        nib.next=a.next
        a.next=nib

    def deleation_at_beagning(self):
        print()
        a=self.head
        self.head=a.next
        self.next=None

n1=node(7)
sll=SLL()
sll.head=n1
n2=node(6)
n1.next=n2
n3=node(5)
n2.next=n3
n4=node(4)
n3.next=n4
n5=node(3)
n4.next=n5
n6=node(2)
n5.next=n6
n7=node(1)
n6.next=n7
sll.traversal()
sll.insert_at_beganing(100)
sll.traversal()
sll.insert_at_end(500)
sll.traversal()
sll.insert_at_specific_node(4,75)
sll.traversal()
sll.deleation_at_beagning()
sll.traversal()
```

```
7 6 5 4 3 2 1
100 7 6 5 4 3 2 1
100 7 6 5 4 3 2 1 500
100 7 6 75 5 4 3 2 1 500
7 6 75 5 4 3 2 1 500
```

In [7]:

```
#deletion at end in singly linked list
class node:
    def __init__(self,data):
        self.data=data
        self.next=None

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

    def traversal(self):
        if self.head is None:
            print("Singly LL is empty")
        else:
            a=self.head
            while a is not None:
                print(a.data, end=" ")
                a=a.next

    def insert_at_beganing(self,data):
        print()
        nb=node(data)
        nb.next=self.head
        self.head=nb

    def insert_at_end(self,data):
        print()
        ne=node(data)
        a=self.head
        while a.next is not None:
            a=a.next
        a.next=ne

    def insert_at_specific_node(self,position,data):
        print()
        nib=node(data)
        a=self.head
        for i in range(1,position-1):
            a=a.next
        nib.next=a.next
        a.next=nib

    def deleation_at_beagning(self):
        print()
        a=self.head
        self.head=a.next
        self.next=None

    def deleat_at_end(self):
        print()
        prev=self.head
        a=self.head.next
        while a.next is not None:
            a=a.next
            prev=prev.next
        prev.next=None

n1=node(7)
sll=SLL()
sll.head=n1
n2=node(6)
n1.next=n2
n3=node(5)
n2.next=n3
n4=node(4)
n3.next=n4
n5=node(3)
n4.next=n5
n6=node(2)
n5.next=n6
n7=node(1)
n6.next=n7
sll.traversal()
sll.insert_at_beganing(100)
sll.traversal()
sll.insert_at_end(500)
sll.traversal()
sll.insert_at_specific_node(4,75)
sll.traversal()
sll.deleation_at_beagning()
sll.traversal()
sll.deleat_at_end()
sll.traversal()
```

```
7 6 5 4 3 2 1
100 7 6 5 4 3 2 1
100 7 6 5 4 3 2 1 500
100 7 6 75 5 4 3 2 1 500
7 6 75 5 4 3 2 1 500
7 6 75 5 4 3 2 1
```

In [11]:

```
#deletion at specific node in singly linked list
class node:
    def __init__(self,data):
        self.data=data
        self.next=None

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

    def traversal(self):
        if self.head is None:
            print("Singly LL is empty")
        else:
            a=self.head
            while a is not None:
                print(a.data, end=" ")
                a=a.next

    def insert_at_beganing(self,data):
        print()
        nb=node(data)
        nb.next=self.head
        self.head=nb

    def insert_at_end(self,data):
        print()
        ne=node(data)
        a=self.head
        while a.next is not None:
            a=a.next
        a.next=ne

    def insert_at_specific_node(self,position,data):
        print()
        nib=node(data)
        a=self.head
        for i in range(1,position-1):
            a=a.next
        nib.next=a.next
        a.next=nib

    def deleation_at_beagning(self):
        print()
        a=self.head
        self.head=a.next
        self.next=None

    def deleat_at_end(self):
        print()
        prev=self.head
        a=self.head.next
        while a.next is not None:
            a=a.next
            prev=prev.next
        prev.next=None

    def deleat_at_specific_node(self,position):
        print()
        prev=self.head
        a=self.head.next
        for i in range(1,position-1):
            a=a.next
            prev=prev.next
        prev.next=a.next
        a.next=None

n1=node(7)
sll=SLL()
sll.head=n1
n2=node(6)
n1.next=n2
n3=node(5)
n2.next=n3
n4=node(4)
n3.next=n4
n5=node(3)
n4.next=n5
n6=node(2)
n5.next=n6
n7=node(1)
n6.next=n7
sll.traversal()
sll.insert_at_beganing(100)
sll.traversal()
sll.insert_at_end(500)
sll.traversal()
sll.insert_at_specific_node(4,75)
sll.traversal()
sll.deleation_at_beagning()
sll.traversal()
sll.deleat_at_end()
sll.traversal()
sll.deleat_at_specific_node(3)
sll.traversal()
```

```
7 6 5 4 3 2 1
100 7 6 5 4 3 2 1
100 7 6 5 4 3 2 1 500
100 7 6 75 5 4 3 2 1 500
7 6 75 5 4 3 2 1 500
7 6 75 5 4 3 2 1
7 6 5 4 3 2 1
```

Doubly linked list

In [27]:

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

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

    def forward_traversal(self):
        if self.head is None:
            print("Doubly LL is empty")
        else:
            a=self.head
            while a is not None:
                print(a.data, end=" ")
                a=a.next

    def backwared_traversal(self):
        print()
        if self.head is None:
            print("Doubly LL is empty")
        else:
            a=self.head
            while a.next is not None:
                a=a.next
            while a is not None:
                print(a.data, end=" ")
                a=a.prev

    def insert_at_beganing(self,data):
        print()
        ns=node(data)
        a=self.head
        a.prev=ns
        ns.next=a
        self.head=ns

    def insert_at_end(self,data):
        print()
        ne=node(data)
        a=self.head
        while a.next is not None:
            a=a.next
        a.next=ne
        ne.prev=a

    def insert_at_specific_node(self,position,data):
        print()
        nib=node(data)
        a=self.head
        for i in range(1,position-1):
            a=a.next
        nib.prev=a
        nib.next=a.next
        a.next.prev=nib
        a.next=nib

    def deleation_at_beagning(self):
        print()
        a=self.head
        self.head=a.next
        a.next=None
        self.head.prev=None

    def deleat_at_end(self):
        print()
        prev=self.head
        a=self.head.next
        while a.next is not None:
            a=a.next
            prev=prev.next
        prev.next=None
        a.prev=None

    def deleat_at_specific_node(self,position):
        print()
        a=self.head.next
        b=self.head
        for i in range(1,position-1):
            a=a.next
            b=b.next
        b.next=a.next
        a.next.prev=b
        a.next=None
        a.prev=None

n1=node(7)
dll=DLL()
dll.head=n1
n2=node(6)
n2.prev=n1
n1.next=n2
n3=node(5)
n3.prev=n2
n2.next=n3
n4=node(4)
n4.prev=n3
n3.next=n4
n5=node(3)
n5.prev=n4
n4.next=n5
n6=node(2)
n6.prev=n5
n5.next=n6
n7=node(1)
n7.prev=n6
n6.next=n7
dll.forward_traversal()
dll.backwared_traversal()
dll.insert_at_beganing(10)
dll.forward_traversal()
dll.insert_at_end(50)
dll.forward_traversal()
dll.insert_at_specific_node(5,500)
dll.forward_traversal()
dll.deleation_at_beagning()
dll.forward_traversal()
dll.deleat_at_end()
dll.forward_traversal()
dll.deleat_at_specific_node(4)
dll.forward_traversal()
```

```
7 6 5 4 3 2 1
1 2 3 4 5 6 7
10 7 6 5 4 3 2 1
10 7 6 5 4 3 2 1 50
10 7 6 5 500 4 3 2 1 50
7 6 5 500 4 3 2 1 50
7 6 5 500 4 3 2 1
7 6 5 4 3 2 1
```

In [ ]:

```
```