LTCE

Experiment No. 5

Aim - Implement various data structures in python & peform their inserting & deleting function.

A 1) WAP to implement singly linked list

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

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

    def listprint(self):
        printvalue = self.head
        while printvalue is not None
            print(printvalue.data)
            printvalue = printvalue.next

list = SL()
list.head = self.head
list.head = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")
list.head.next = e2
e2.next = e3
list.listprint()
```

⇒ OP ⇒ Mon
Tue
Wed

A 2) WAP to insert a element in the linked list

Program -

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

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

    def printlist(self):
        value = self.head
        while value is not None:
            print(value.data)
            print()
            value = value.next

    def AddElement(self, newdata):
        NewNode = Node(newdata)
        NewNode.next = self.head
        self.head = NewNode

list = SL()
```

```
list.head = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")

list.head.next = e2
e2.next = e3
list.AddElement("sun")
list.printlist()
```

⇒ Output ⇒ Sun
Mon
Tue
Wed

A 3) WAP to remove element from linked list.

Program -

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

class SL:
    def __init__(self):
        self.head = None
    def Add(self, datain):
        NewNode = Nota(datain)
        NewNode.next = self.head
        self.head = NewNode
```

```
def RemoveNode(self, remove):
    Head = self.head
    if (Head is Not None):
        if (Head.data == remove):
            self.head = Head.next
            Head = None
            return
    while (Head is not None):
        if Head.data = remove:
            break
        prev = Head
        Head = Head.next
    if (Head == None):
        return
    prev.next = Head.next
    Head = None

def Lprint(self):
    value = self.head
    while (value):
        print(value.data),
        value = value.next

llist = SLL()
llist.Add("Mon")
llist.Add("Tue")
llist.Add("Wed")
llist.Add("Thu")
llist.RemoveNode("Tue")
llist.Lprint()
```

Output ⇒ Thu
Wed
Mon

8) WAP to implement stack & its push() & pop() function.

Program -

```
def createStack():
    stack = []
    return stack

def checkEmpty(stack):
    return len(stack) == 0

def push(stack, item):
    stack.append(item)
    print("pushed item: " + item)

def pop(stack):
    if (checkEmpty(stack)):
        return "stack is empty"
    return stack.pop()

stack = createStack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
```

```
print("stack before popping an element:" + str(stack))
print("popped item:" + pop(stack))
print("stack after popping an element:" +str(stack))
```

Output ⇒ pushed item: 1
pushed item: 2
pushed item: 3
pushed item: 4
Stack before popping an element: ['1', '2', '3', '4']
popped item: 4
Stack after popping an element: ['1', '2', '3']

() WAP to implement Queue & its push & pop functions

```
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)\n print("Before removing element")\n
q.display()\n print(
q.dequeue()
print("After removing an element")
q.display()
```

Output ⇒ Before removing an element
[1, 2, 3, 4, 5]
After removing an element
[2, 3, 4, 5]