Practical No. 7

Aim : To study Queue add and delete in python.

Theory : A Queue is linear structure which follows a particular order in which operation are performed in the order in first in first out.

A good example of a queue is any queue of consumers for a resource where the consumer that was added to the queue first is served first.

Application of Queue : Queue is used when things don't have to be processed immediately but have to be processed in first in first out order like this property of Queue make it useful in following kind of scenarios.

1) When a resource is shared among multiple consumer example include CPU scheduling, Disk scheduling.

2) When data is transferred asynchronously between two processes. Example include to Buffer a file

```
## Queue add and Delete ##
Print("Name-Mayuresh Rane" )
Print("roll no. 1713")


class Queue:
  global r
  global f
  def __init__(self):
    self.r=0
    self.f=0
    self.l=[0,0,0,0,0,0]
  def add(self,data):
    n=len(self.l)
    if self.r<n-1:
      self.l[self.r]=data
      self.r=self.r+1
    else:
      print("Queue is full")
  def remove(self):
    n=len(self.l)
    if self.f<n-1:
      print(self.l[self.f])
      self.f=self.f+1
    else:
      print("Queue is empty")
Q=Queue()
Q.add(30)
Q.add(40)
Q.add(50)
Q.add(60)
Q.add(70)
```

```
Q.add(80)


Q.remove()
Q.remove()
Q.remove()
Q.remove()
Q.remove()
Q.remove()
```

```
print("NAME : Mayuresh
class node:
    global data
    global next
    def __init__(self,item):
        self.data=item
        self.next=None
class linkedlist:
    global s
    def __init__(self):
        self.s=None
    def addL(self,item):
        newnode=node(item)
        if self.s==None:
            self.s=newnode
        else:
         head=self.s
         while head.next!=None:
            head=head.next
         head.next=newnode
    def addB(self,item):
        newnode=node(item)
        if self.s==None:
            self.s=newnode
        else:
            newnode.next=self.s
            self.s=newnode
    def display(self):
         head=self.s
         while head.next!=None:
            print (head.data)
            head=head.next
         print (head.data)
start=linkedlist()
start.addL(13)
start.addL(85)
start.addL(90)
start.addL(88)
start.addB(44)
start.addB(46)
start.addB(29)
start.display()

---OUTPUT---
NAME : Mayuresh Rane
ROLL NO : 1713
29
46
44
88
90
85
13
```

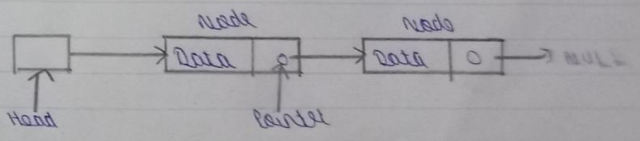
Practical-8 042

Aim :- To demonstrate the use of linked list in data structure

Theory :- A linked list is a sequence of data structure. Linked list is a sequence of links which contains items. Each link contains a connection to another link.

- LINK - Each link of linked list can store a data called an element.
- NEXT - Each link of a linked list contains a link to the next link called NEXT.
- LINKED LIST - A linked list contains the connection link to the first link called first.

LINKED LIST ~~Representation~~ :



```
---PROGRAM---
print("NAME : Mayuresh Rane \nROLL NO : 1713")
def evaluate(s):
    k=s.split()
    n=len(k)
    stack=[]
    for i in range(n):
        if k[i].isdigit():
            stack.append(int(k[i]))
        elif k[i]=='+':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)+int(a))
        elif k[i]=='-':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)-int(a))
        elif k[i]=='*':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)*int(a))
        else:
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)/int(a))
    return stack.pop()
s="4 6 8 * + "
r=evaluate(s)
print("The evaluated value is:",r)

---OUTPUT---
NAME : Mayuresh Rane
ROLL NO : 1713
The evaluated value is: 52
```

Practical - 9 C43

Aim :- To evaluate postfix expression using stack

Theory :- Stack is an (ADT) and work on LIFO i.e PUSH and POP operation

A postfix expression is a collection of operators and operands in which the operator is placed after the operands

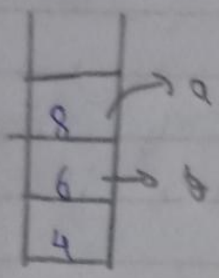
Steps to be followed :-

- Read all the symbols one by one from left to right in given postfix expression
- If the reading symbol is operand then push it on to the stack
- If the reading symbol is operator (+, -, *, /) the perform TWO pop operations and store the two popped operands in two different variables (operand1 & operand2)
- Then perform reading symbol operation using operand1 & operand2 and push result back on to the stack.
- Finally, perform a POP operation and display the popped value as final result.

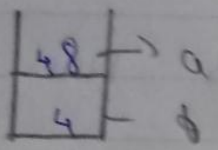
#87

value of postfix expression :

S = " 4 6 8 * + "



b * a = 6 × 8 = 48 ll store again in stack



a + a = 4 + 48 = 52

```
##Quick sort##
Print("Mayuresh Rane")
Print("Roll no.--1713")
def quickSort(alist):
  quickSortHelper(alist,0,len(alist)-1)
def quickSortHelper(alist,first,last):
  if first<last:
    splitpoint=partition(alist,first,last)
    quickSortHelper(alist,first,splitpoint-1)
    quickSortHelper(alist,splitpoint+1,last)
def partition(alist,first,last):
  pivotvalue=alist[first]
  leftmark=first+1
  rightmark=last
  done=False
  while not done:
    while leftmark<=rightmark and alist[leftmark]<=pivotvalue:
      leftmark=leftmark+1
    while alist[rightmark]>=pivotvalue and rightmark>=leftmark:
      rightmark=rightmark-1
    if rightmark<leftmark:
      done=True
    else:
      temp=alist[leftmark]
      alist[leftmark]=alist[rightmark]
      alist[rightmark]=temp
  temp=alist[first]
  alist[first]=alist[rightmark]
  alist[rightmark]=temp
  return rightmark
alist=[42,54,45,67,89,66,55,80,100]
quickSort(alist)
print(alist)
```

Output:
Mayuresh rane
Roll no.--1713

[42,45,54,55,66,89,67,80,100]

051

Practical - 10

Aim :- To 'sort' i.e to sort the given data in Quick sort.

Theory :- Quicksort is an efficient sorting algorithm. Type of a divide & conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

- There are many different versions of quick sort that pick pivot in different ways.

1) Always pick first element as pivot.
2) Always pick last element as pivot.
3) pick a random element as pivot.
4) Pick median as pivot.

- The key process in quicksort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array & put all smaller elements before x, & put all greater elements after x All this should be done in linear time.

M/E.

Aim : Binary Tree and Traversal

Theory : A tree can have a number of children depending on the maximum number of children a tree supports. The top of the tree may vary widely from another tree.

Binary tree is a tree which supports maximum of 2 children for any node within the tree. Thus any particular node can have either 0 or 1 or 2 children.

There is another identity of binary tree that it is ordered such that one child is identified as left child and other as right child.

```
## Binary Tree and Traversal ##
Print("Name:Mayuresh Rane)
Print("roll:1713")

class Node:
  global r
  global l
  global data
  def __init__(self,l):
    self.l=None
    self.data=l
    self.r=None
class Tree:
  global root
  def __init__(self):
    self.root=None
  def add(self,val):
    if self.root==None:
      self.root=Node(val)
    else:
      newnode=Node(val)
      h=self.root
      while True:
        if newnode.data < h.data:
          if h.l!=None:
            h=h.l
          else:
            h.l=newnode
            print(newnode.data,"added on left of",h.data)
            break
        else:
          if h.r!=None:
            h=h.r
          else:
            h.r=newnode
            print(newnode.data,"added on right of",h.data)
            break
  def preorder(self,start):
    if start!=None:
      print(start.data)
      self.preorder(start.l)
      self.preorder(start.r)
  def inorder(self,start):
    if start!=None:
      self.inorder(start.l)
      print(start.data)
      self.inorder(start.r)
  def postorder(self,start):
    if start!=None:
      self.inorder(start.l)
      self.inorder(start.r)
      print(start.data)
T=Tree()
T.add(100)
T.add(80)
T.add(70)
T.add(85)
T.add(10)
T.add(78)
T.add(60)
T.add(88)
T.add(15)
```

- Size of a Binary tree is the number of nodes in the tree.

```
T.add(12)
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
```

Output:

Name:Mayuresh Rane

Roll1713

```
80 added on left of 100
70 added on left of 80
85 added on right of 80
10 added on left of 70
78 added on right of 70
60 added on right of 10
88 added on right of 85
15 added on left of 60
12 added on left of 15
preorder
100
80
70
10
60
15
12
78
85
88
inorder
10
12
15
60
70
78
80
85
88
100
postorder
10
12
15
60
70
78
80
85
88
100
```

Practical No :- 12

Aim :- Merge Sort

Theory :- Merge sort is a sorting technique based on divide and conquer technique with worst-case time complexity being O (n log n). it is one of the most respected algorithm.

Merge sort first divides the array into equal halfs and then combines them in a sorted manner.

It divides input array in two halves, calls itself for the two halves and then merge the two sorted halves. The merge () is used for merging 2 halves. The merge (arr, l, m, r) is key process that assume that arr [l ... m] and arr [m+1 ... r] are sorted and merges the two sorted sub-array in one.


```
##MERGE SORT##
Print("Name:Mayuresh rane")
Print("roll: 1713")


def sort(arr,l,m,r):
    n1=m-l+1
    n2=r-m
    L=[0]*(n1)
    R=[0]*(n2)
    for i in range(0,n1):
        L[i]=arr[l+i]
    for j in range(0,n2):
        R[j]=arr[m+1+j]
    i=0
    j=0
    k=l
    while i<n1 and j<n2:
        if L[i]<=R[j]:
            arr[k]=L[i]
            i+=1
        else:
            arr[k]=R[j]
            j+=1
            k+=1
        while i<n1:
            arr[k]=L[i]
            i+=1
            k+=1
        while j<n2:
            arr[k]=R[j]
            j+=1
            k+=1
    def mergesort(arr,l,r):
        if l<r:
            m=int((l+(r-1))/2)
            mergesort(arr,l,m)
            mergesort(arr,m+1,r)
            sort(arr,l,m,r)
    arr=[12,23,34,56,78,45,86,98,42]
    print(arr)
    n=len(arr)
    mergesort(arr,0,n-1)
    print(arr)

    Output:
    Name:Mayuresh rane
    Roll:1713
```

```
[12, 23, 34, 56, 78, 45, 86, 98, 42]
[12, 23, 56, 56, 42, 45, 78, 86, 98]
>>>
```