# PRACTICAL No. 1.

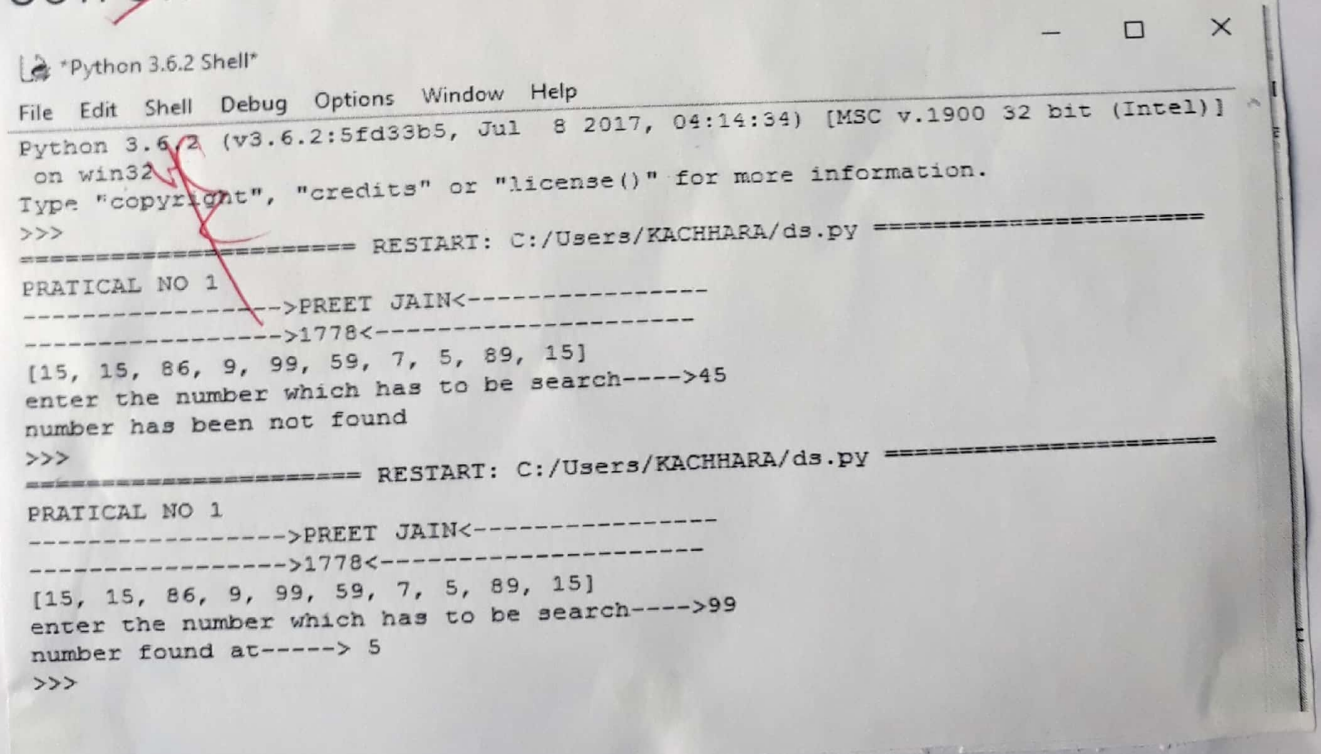
Aim: To search a number from the list using linear unsorted

Theory: The process of identifying or finding a particular record is called searching.
There are two types of searching: i) Linear Search
ii) Binary Search.

The linear is further classified as
i) Sorted ii) Unsorted
Here we will look on the UNSORTED linear search.
linear search, also known as sequential search, is a process that checks every element in the list sequentially until the desired element is found. when the element to be searched are not specifically arranged in ascending or descending order. They are arranged in random manner. That is what it calles unsorted linear search.

```
print("PRATICAL NO 1")
print("---------->PREET JAIN<----------")
print("---------->1778<----------")
a=[15,15,86,9,99,59,7,5,89,15]
j=0
print(a)
search=int(input("enter the number which has to be search---->"))
for i in range(len(a)):
  if (search==a[i]):
    print("number found at----->",i+1)
    j=1
    break
if(j==0):
  print("number has been not found ")
```

# OUTPUT:



```
*Python 3.6.2 Shell*
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/KACHHARA/ds.py ==========
PRATICAL NO 1
---------->PREET JAIN<----------
---------->1778<----------
[15, 15, 86, 9, 99, 59, 7, 5, 89, 15]
enter the number which has to be search---->45
number has been not found
>>>
========== RESTART: C:/Users/KACHHARA/ds.py ==========
PRATICAL NO 1
---------->PREET JAIN<----------
---------->1778<----------
[15, 15, 86, 9, 99, 59, 7, 5, 89, 15]
enter the number which has to be search---->99
number found at-----> 5
>>>
```

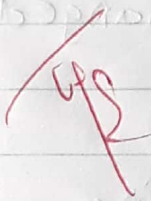
Unsorted linear search

-> The data is entered in random manner.

-> User needs to specify the element to the searched in the entered list.

-> Check the condition that whether the entered number matches if it matches then display the location plus increment 1 as data is stored from location zero.

-> If all elements are checked one by one and element not found then prompt message number not found.



AE

# PRACTICAL 2

Aim: To search a number from the list using linear sorted method.

Theory: SEARCHING AND SORTING.
are different modes or types of data-structure
SORTING: To basically sort the input data in ascending or descending order.
SEARCHING - To search elements and display the same.

In Searching that too in LINEAR SORTED search the data is arranged in ascending to descending or descending to ascending that is all what it mean by searching through 'sorted' that is well arranged data.

# SOURCE CODE:

```
print("PRATICAL NO 2")
print("---------->PREET JAIN<----------")
print("---------->1778<----------")
a=[19,15,84,29,16,32,46,5,8]
j=0
print(a)
search=int(input("enter the number which has to be search---->"))
if ((search<a[0]) or (search>a[6])):
   print("given number doesnt exist!")
else:
   for i in range(len(a)):
      if (search==a[i]):
         print("number found at----->",i+1)
         j=1
         break
   if(j==0):
      print("number has been not found ")
```

# output:

```
Python 3.6.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/KACHHARA/ds.py ==========
PRATICAL NO 2
---------->PREET JAIN<----------
---------->1778<----------
[19, 15, 84, 29, 16, 32, 46, 5, 8]
enter the number which has to be search---->4
given number doesnt exist!
>>>
========== RESTART: C:/Users/KACHHARA/ds.py ==========
PRATICAL NO 2
---------->PREET JAIN<----------
---------->1778<----------
[19, 15, 84, 29, 16, 32, 46, 5, 8]
enter the number which has to be search---->31
number has been not found
>>>
```

Sorted linear search.

-> The user is supposed to enter data in sorted manner.

->User has to given an element for search through sorted list.

-> If element is found display with an updation as value is sorted from location '0'.

-> If data or element not found print the same.

-> In sorted order list of elements we can check the condition that whether the entered number lies from starting point till the last element if not then without any processing we can say number is not in the list.

# PRACTICAL-3.

Aim: To search a number from the given sorted list using binary search.

Theory: A linear search also known as a half-interval search. is an algorithm used in computer science to locate a specified value (key) within the array.

For the search to the binary. the array must be sorted in either ascending or descending order.

At each step of the algorithm a comparision is made and a procedure branches into one of two directions.

Specifically, the key value is less than or greater than this middle element. the algorithm knows which half of the array to continue searching in because the array is sorted.

This process is repeated on progressively smaller segments of the array until the value is located.

```
Print("pratical no 3")
print("---------->PREET JAIN<----------")
print("---------->1778<----------")
a=[3,12,23,36,49,66,78]
print(a)
search=int(input("Enter number to be searched from the list:")
l=0
h=len(a)-1
m=int((l+h)/2)
if((search<a[l]) or (search>a[h])):
   print("Number not in RANGE!")
elif(search==a[h]):
   print("number found at location :",h+1)
elif(search==a[l]):
   print("number found at location :",l+1)
else:
   while(l!=h):
      if(search==a[m]):
         print ("Number found at location:",m+1)
         break
      else:
         if(search<a[m]):
            h=m
            m=int((l+h)/2)
         else:
            l=m
            m=int((l+h)/2)
      if(search!=a[m]):
         print("Number not in  given list!")
         break
```

Python 3.6.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/KACHHARA/AppData/Local/Programs/Python/Python36-32/preet.py
pratical no 3
-------------->PREET JAIN<------------------
----------------------->1778<-----------
[3, 12, 23, 36, 49, 66, 78]
Enter number to be searched from the list:49
Number found at location: 5
>>>
 RESTART: C:/Users/KACHHARA/AppData/Local/Programs/Python/Python36-32/preet.py
pratical no 3
-------------->PREET JAIN<------------------
----------------------->1778<-----------
[3, 12, 23, 36, 49, 66, 78]
Enter number to be searched from the list:99
Number not in RANGE!
>>>
 RESTART: C:/Users/KACHHARA/AppData/Local/Programs/Python/Python36-32/preet.py
pratical no 3
-------------->PREET JAIN<------------------
----------------------->1778<-----------
[3, 12, 23, 36, 49, 66, 78]
Enter number to be searched from the list:11
Number not in  given list!
>>>
```

Because each step in the algorithm divides the array size in half. a linary search will complete successfully in logarithmic time.

B4 PRACTICAL No. 6

Aim: To demonstrate the use of stack

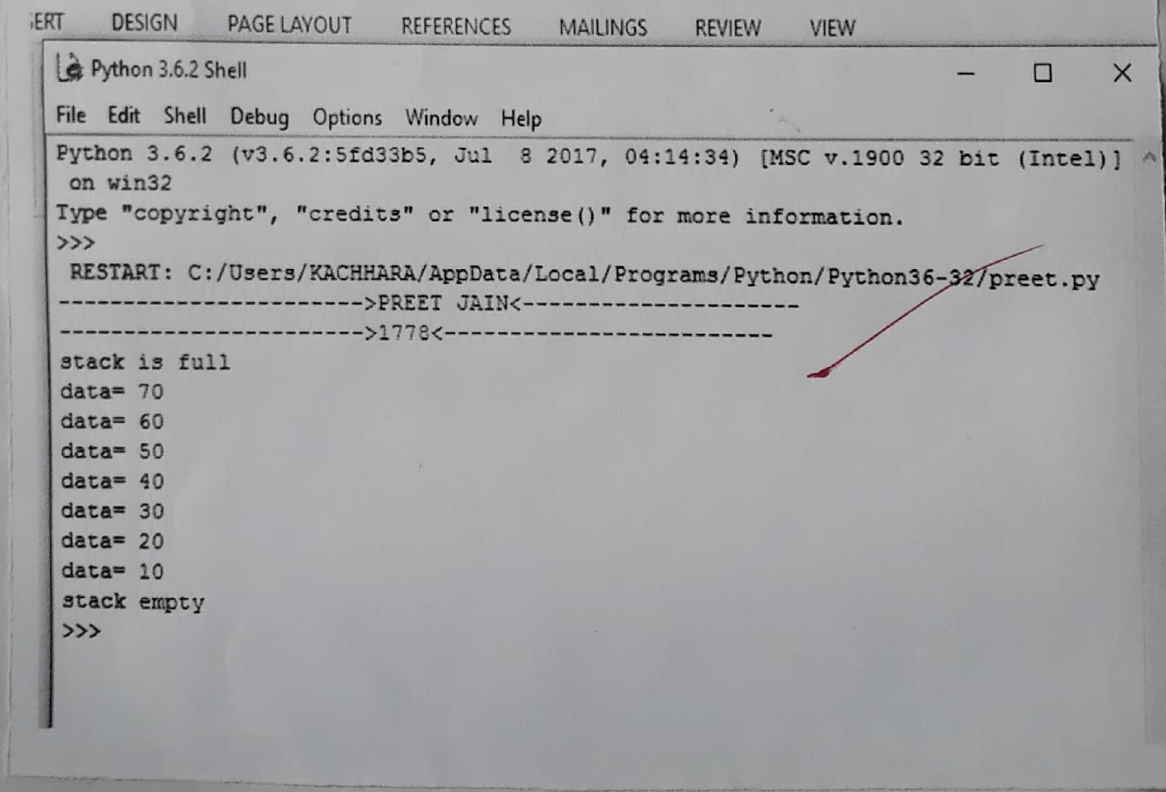
Theory: In computer science, a stack is an abstract data type that serves as a collection of elements with two principal operations push, which adds an element to the collection and pop, which removes the most recently added element that was not yet removed.
The order may be LIFO (Last In First Out) or FILO (First In Last Out)
There basic operation are performed in the stack.

- Push:
  Adds an item in the stack if the stack is full then it is said to be overflow condition.
- Pop:
  Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an underflow condition.

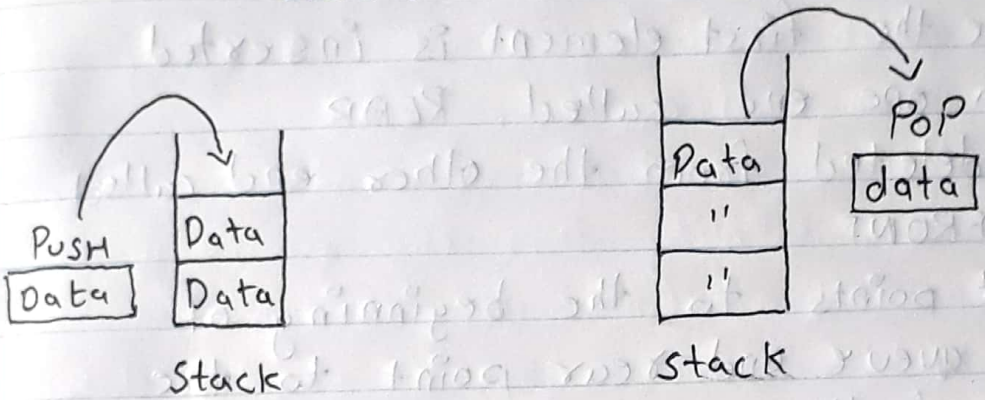
```
print("---------------------->PREET JAIN<----------------------")
print("---------------------->1778<----------------------")
class stack:
    global tos
    def __init__(self):
        self.l=[0,0,0,0,0,0,0]
        self.tos=-1
    def push(self,data):

        n=len(self.l)
        if self.tos==n-1:
            print("stack is full")
        else:
            self.tos=self.tos+1
            self.l[self.tos]=data
    def pop(self):
        if self.tos<0:
            print("stack empty")
        else:
            k=self.l[self.tos]
            print("data=",k)
            self.tos=self.tos-1
s=stack()
s.push(10)
s.push(20)
s.push(30)
s.push(40)
s.push(50)
s.push(60)
```

```
s.push(70)
s.push(80)
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
```

OUTPUT:



- Peek or Top : Returns top element of stack
- is Empty : Returns true if stack is empty else false.




Last-in-First-out

# PRACTICAL No. 5

AIM: To demonstrate Queue add and delete

THEORY: Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other end called as FRONT

Front points to the beginning of the queue and rear point to the end of the queue.

Queue follows the FIFO (First in First out) structure

According to its FIFO structure element inserted first will also be removed first.

In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue) because queue is open at both of its end.

enqueue() can be termed as add() in queue i.e adding a element in queue.

Dequeue() can be termed as delete or remove i.e. deleting or removing of element.

```
print("---------->PREET JAIN<----------")
print("---------->1778<----------")
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()

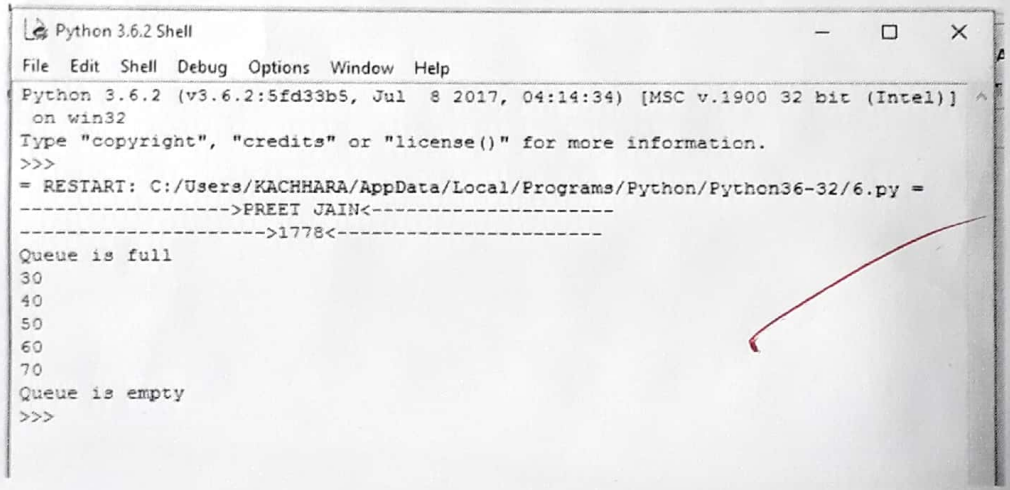
Q.remove()

Q.remove()

Q.remove()

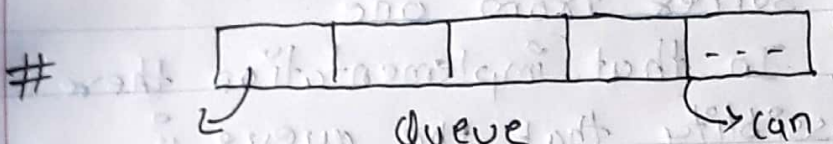
Q.remove()

OUTPUT :

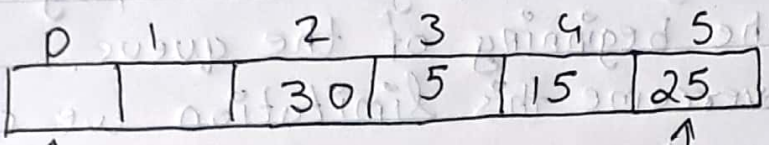


Front is used to get the front data item from a queue.

Rear is used to get the last item from a queue.




on both sides




Front = 2
Rear = 5

Front

Rear

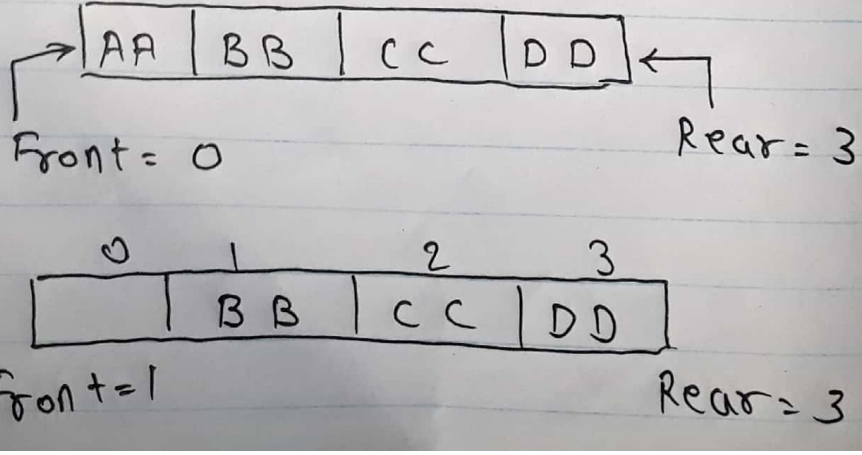
# PRACTICAL No. 7.

Aim: To demonstrate the use of circular queue in data-structure.

THEORY:

The queue that use implement using an array suffer from one limitation. In that implementation there is a possiblity that the queue is reported as full, even though is actuality there might be empty slots at the beginning of the queue. To overcome this limitation we can implement queue as circular queue. In circular queue we go on adding the element to the queue and reach the end of the array. The next element is stored in the first slot of the array.

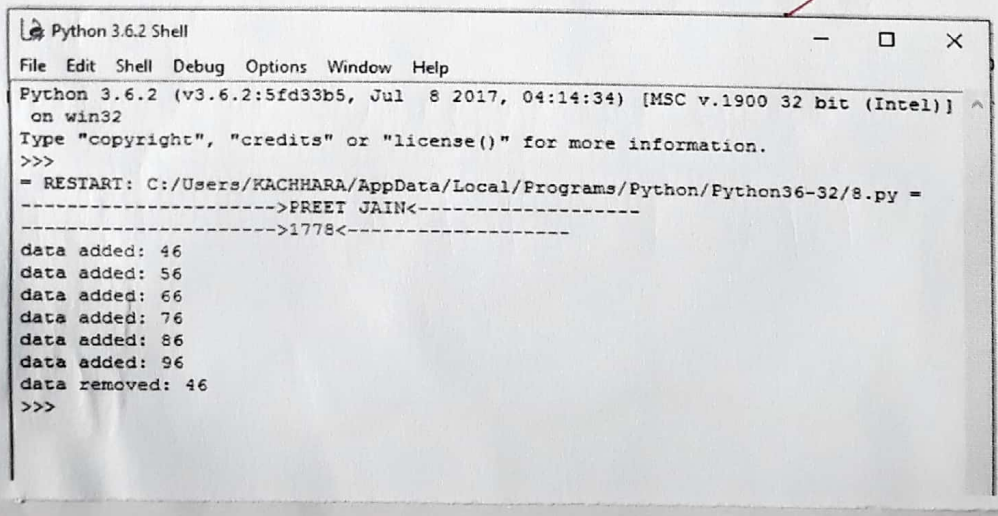
Example:

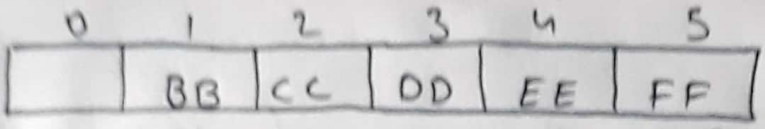


```
print("---------->PREET JAIN<----------")
print("---------->1778<----------")
class Queue:
  global r
  global f
  def __init__(self):
    self.r=0
    self.f=0
    self.l=[0,0,0,0,0,0]
  def add(self,data):
    n=len(self.l)
    if self.r<=n-1:
      self.l[self.r]=data
      print("data added:",data)
      self.r=self.r+1
    else:
      s=self.r
      self.r=0
      if self.r<self.f:
        self.l[self.r]=data
        self.r=self.r+1
      else:
        self.r=s
        print("Queue is full")
  def remove(self):
    n=len(self.l)
    if self.f<=n-1:
      print("data removed:",self.l[self.f])
      self.f=self.f+1
```

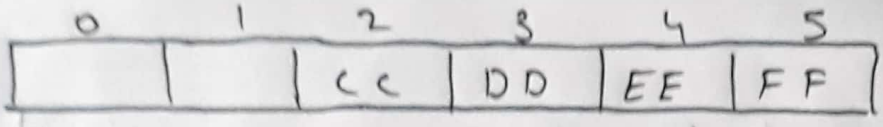
```
        else:
            s=self.f
            self.f=0
            if self.f<self.r:
                print(self.l[self.f])
                self.f=self.f+1
            else:
                print("Queue is empty")
                self.f=s
Q=Queue()
Q.add(46)
Q.add(56)
Q.add(66)
Q.add(76)
Q.add(86)
Q.add(96)
Q.remove()
Q.add(76)
```

OUTPUT:

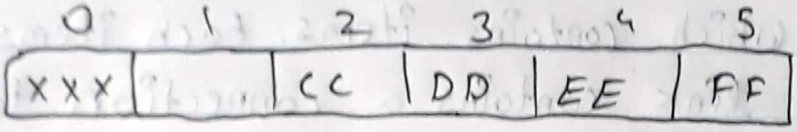





Front=1 Rear=5




Front=2 Rear=5




Front=2 Rear=0

P.P

# PRACTICAL NO. 8

Aim:

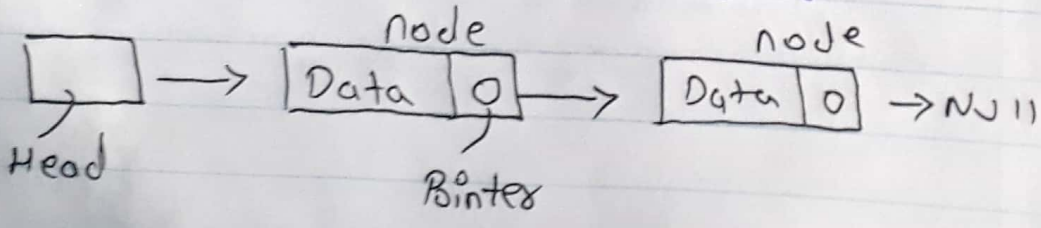
To demonstrate the use of linked list in data structure.

Theory:

A linked list is a sequence of data structures. Linked list is a sequence of links which contains items. Each items each link contains a connection to another link.

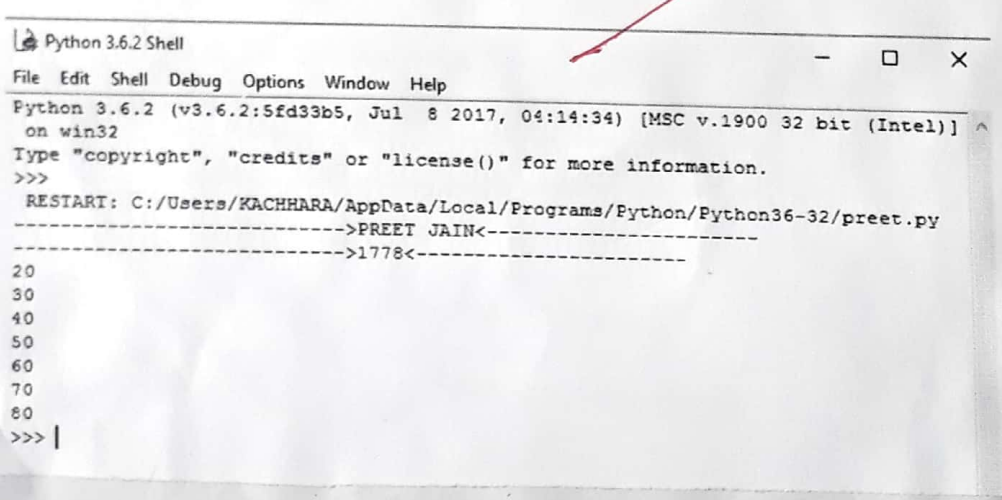
- LINK: Each link of a linked list can store a data called an element
- NEXT: Each link of a linked list contains a link to the next link called NEXT.
- LINKED LIST: A linked list contains the connection link to the first link called First.

LINKED LIST REPRESENTATION:



```
print("---------->PREET JAIN<----------")
print("---------->1778<----------")
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(50)
start.addL(60)
start.addL(70)
start.addL(80)
start.addB(40)
start.addB(30)
start.addB(20)
start.display()
```

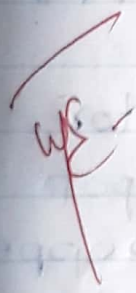


Types of Linked list :

→ Simple
→ Doubly
→ Circular

Basic Operations

→ Insertion
→ Deletion
→ Display
→ search
→ Delete



# PRACTICAL- 9

Aim: To evaluate postfix expression using stack.

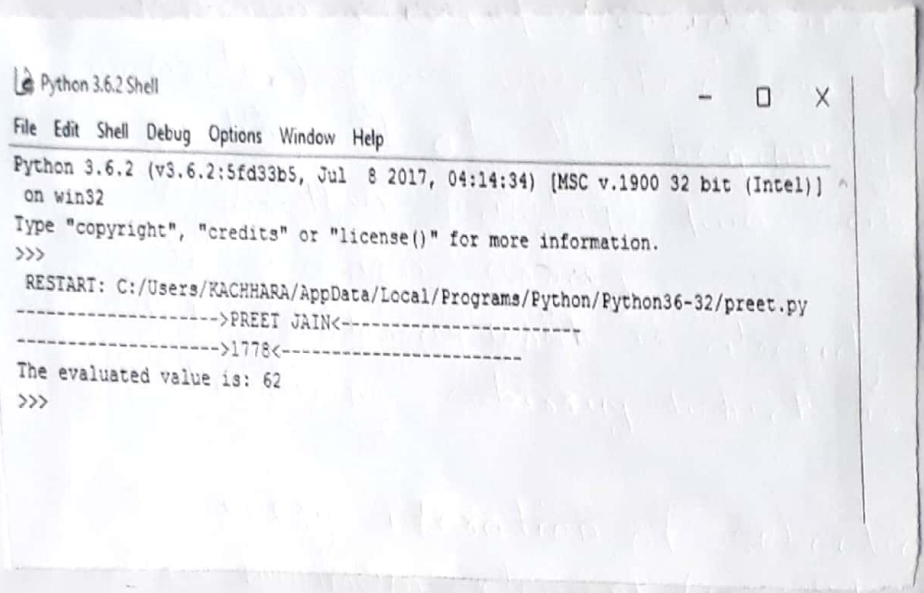
Theory:

Stack is an ADT and works on LIFO (Last-in-first-out) i.e. PUSH & POP Operation. A postfix expression is a collection of operator and operands in which the operator is placed after the operands.

STEPS TO FOLLOW:

1) Read all the symbol one by one from left to right in the given postfix expression.
2) In the reading symbol is operand then push it on to the stack.
3) If the reading symbol is operator (+, −, *, /, etc) then perform Two pop operations and store the two popped operands in two different variable (operand 1 & operand 2) Then perform reading sysmbol operator using operand) & operand 2 and push reult back on to the stack
4) Finally! Perform a pop operation and display the popped valve as final result.

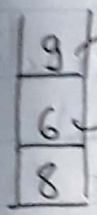
```
print("---------->PREET JAIN<----------")
print("---------->1778<----------")
def evaluate(s):
  k=s.split()
  n=len(k)
  stack=[]
  for i in range(n):
    if k[i].isdigit():
      stack.append(int(k[i]))
    elif k[i]=='+':
      a=stack.pop()
      b=stack.pop()
      stack.append(int(b)+int(a))
    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="8 6 9 * +"
r=evaluate(s)
print("The evaluated value is:",r)
```



Values of postfix expression.

8 6 9 * +

Stack :




b * a = 9 * 6 = 54 // store again is stack


b + a = 8 + 54
= 62 //

PAGE NO.

# PRACTICAL No. 10 (A)

## SELECTION SORT

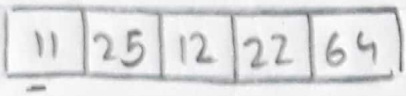
THEORY :

The Selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

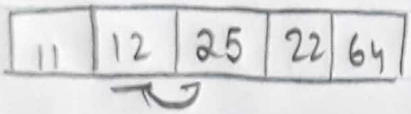
1) The subarray which is already sorted
2) Remaining subarray which is unsorted

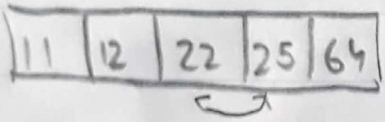
In every iteration of selection sort, the minimum element (considering ascending order) from unsorted subarray is picked and remove to sorted array

eg->




Unsorted

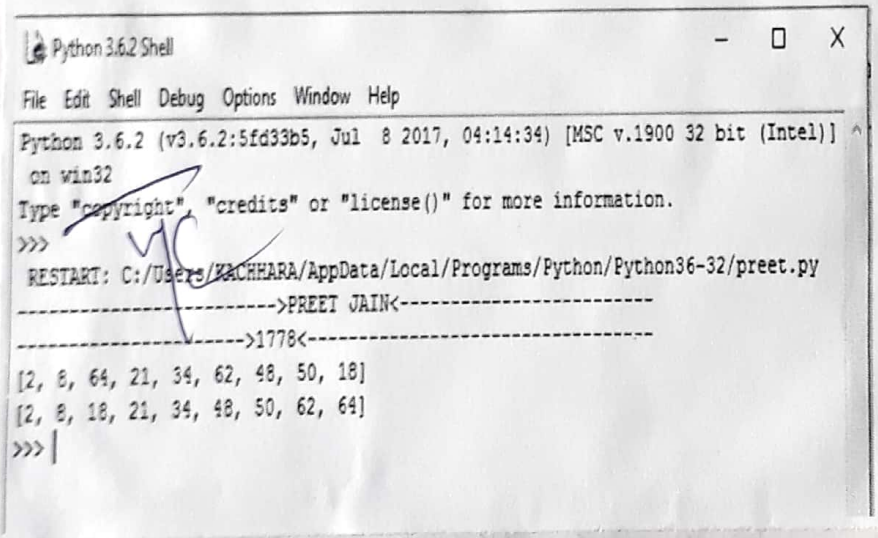



Sorted

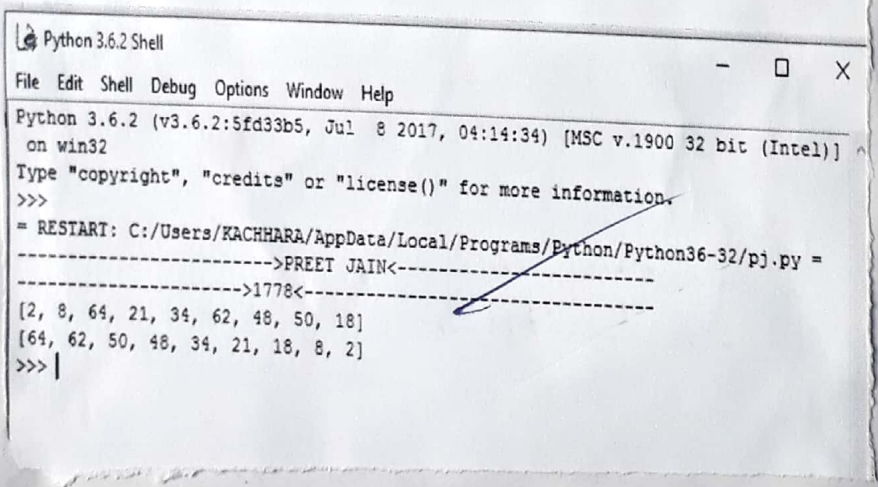
```
#selection sort#
print("---------------------->PREET JAIN<----------------------")
print("---------------------->1778<----------------------------")
a=[2,8,64,21,34,62,48,50,18]
print(a)
for i in range(len(a)-1):
   for j in range(len(a)-1):
      if(a[j]>a[i+1]):
         t=a[j]
         a[j]=a[i+1]
         a[i+1]=t
print(a)
```

# OUTPUT:



```
#bubble sort
print("---------------------->PREET JAIN<----------------------")
print("---------------------->1778<----------------------------")
a=[2,8,64,21,34,62,48,50,18]
print(a)
for i in range(len(a)-1):
    for j in range(len(a)-1-i):
        if(a[j]<a[j+1]):
            t=a[j]
            a[j]=a[j+1]
            a[j+1]=t
print(a)
```

# OUTPUT:

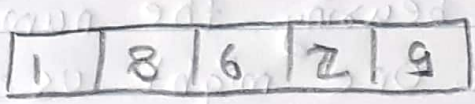


## BUBBLE SORT

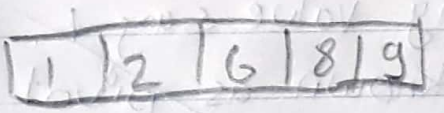
THEORY :

Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated unit the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list.

e.g →




unsorted




sorted

# PRACTICAL NO. 12.

## QUICK SORT:

THEORY: Quick sort is one of the most efficient sorting algorithm and is based on the splitting of an array into smaller ones. The name comes from the fact that, quick sort is capable of sorting a list of data elements significantly faster than any of the common sorting algorithms.

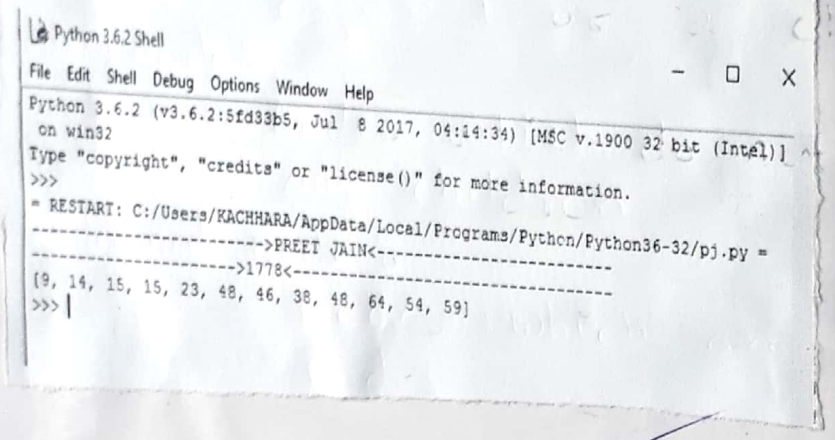
Quick sort works in the following manner

1) Taking the analogical view in perspective, consider a situation where one had to sort the papers bearing the names of the students, by name. One might use the approach as follows:
   a) Select a splitting value say, L. The splitting value is also known as pivot
   b) divide the stack of paper into A-L and M-Z. It is not necessary that the piles should be equal.
   c) Repeat the above two steps with A-L pile splitting it into its significant two halves
   d) Ultimately, the smaller piles can be placed one on top of the other to

```
print("----------->PREET JAIN<-----------")
print("----------->1778<-----------")
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=[54,15,46,48,23,15,9,14,64,48,38,59]
quickSort(alist)
print(alist)
```

## OUTPUT:



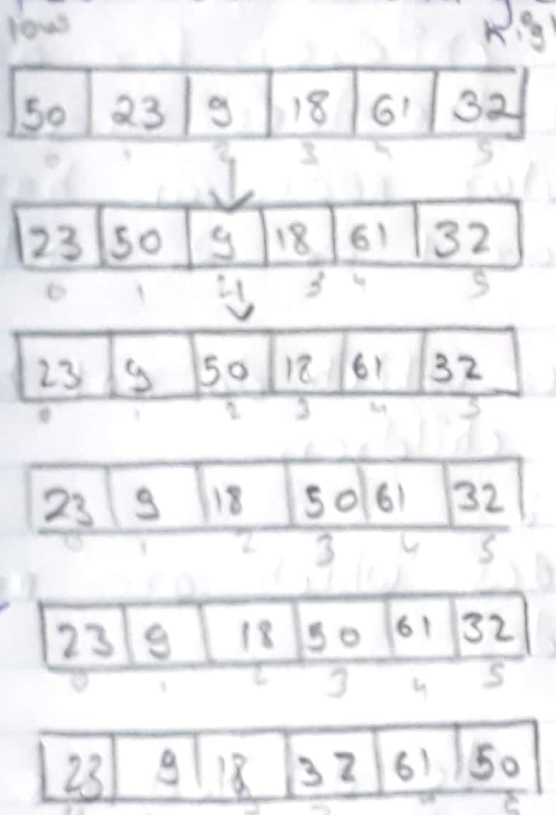
produce a fully sorted and ordered set of papers.

3) The approach used here is recursion at each split to get to the single-element array.

4) At every split, the pile was divided and then the same approach was used for the smaller piles.

5) Due to these features, quick sort is also called partition exchange sort.

eg →



# PRACTICAL No. 13.

Topic : Binary tree.

Theory :

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

deaf node :

Nodes which do not have any children

Internal node :

Nodes which are non-leaf nodes

Traversing can be defined as a process of visiting every node of the tree exactly once

```
print("---------->PREET JAIN<----------")
print("---------->1778<----------")
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.postorder(start.l)
            print(start.data)
            self.postorder(start.r)
```

Preorder:

i) visit the root node.
ii) Tranverse the left subtree. The left subtree in turn might have left and right subtrees.
iii) Transverse the right subtree. The right subtree inturn might have left and right subtree

Inorder:

i) tranverse the left subtree. The left subtree inturn might have left and right subtree.
ii) visit the rood node.
iii) Transverse the right subtree. The right subtree inturn might have left & right subtrees.

Postorder:

i) Traverse the left subtree. The left subtree intern might have left and right subtrees.

ii) Traverse the right subtree. The right subtree intern might have left & right subtrees.

iii) Visit the root node.

```
T=Tree()
T.add(120)
T.add(90)
T.add(40)
T.add(83)
T.add(78)
T.add(111)
T.add(60)
T.add(59)
T.add(152)
T.add(39)
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
```

OUTPUT :

```
Python 3.4.3 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>> ========== RESTART ==========
>>>
------------------------->PREET JAIN<-------------------------
--------------------->1778<---------------------
90 added on left of 120
40 added on left of 90
83 added on right of 40
78 added on left of 83
111 added on right of 90
60 added on left of 78
59 added on left of 60
152 added on right of 120
39 added on left of 40
preorder
120
90
40
39
83
78
60
59
111
152
inorder
39
40
59
60
78
83
90
111
120
152
postorder
39
40
59
60
78
83
90
111
120
152
>>>
```

```
print("---------->PREET JAINN<----------")
print("---------->1778<----------")
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=1
  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]
```

# PRACTICAL No. 14.

Topic : Merge Sort.

Theory :

Merge Sort uses the divide and conquer technique. In merge sort we divide the list into nearly equal sublist each of which are then again divided into two sublists.

- We continue this pocedure until there is only one element in the individual sublist.
- One we have individual element in the sublists, we consider these sublist as an sub problems which can be solved.
  Since ther is only one elements in the sublist, the sublist is already sorted so now we merge the sub problems.
- Now while merging the sub programs, we compare the 2 sublist and create the combined list by comparing

the element of the sublist.
After comparing we place the element in their correct position in the original sublists.

```
        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=[4,25,16,27,48,14,6]
print(arr)
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```

# OUTPUT: