Aniket Prajapati
SYCS –351
DS JOURNAL

PRACTICAL NO – 1

Aim: -

A. Write a program to store the elements in 1-D array and provide an option to perform the operations like searching, sorting, merging, reversing the elements.

Theory: -

Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

Element− Each item stored in an array is called an element.

Index − Each location of an element in an array has a numerical index, which is used to identify the element.

Basic Operations

Following are the basic operations supported by an array.

Traverse − print all the array elements one by one.

Insertion − Adds an element at the given index.

Search − Searches an element using the given index or by the value.

Sorting-Means Arranging the Element in particular order. i.e., Ascending or Descending order.

Aniket Prajapati
SYCS –351
DS JOURNAL

Code: -

Practical 1.py - D:/DS/DS practi/Practical 1.py (3.7.0)

File Edit Format Run Options Window Help

```
# Name  :Aniket Prajapati
# Rollno:351
# Class :SYCS
from array import *
class Stack():
    def __init__(self):
        self.items = array('i',[4,3,2,351])

    def end(self, item):
        self.items.append(item)
        print(item)

    def peek(self):
        if self.items:
            return self.items[-1]
        else:
            return None

    def size(self):
        if self.items:
            return len(self.items)
        else:
            return None

    def display(self):
        for i in self.items:
            print(i)

    def start(self, i):
        self.items.insert(0, i)
#searching
    def search(self, a):
        l = self.items
        for i in l:
            if i == a:
                print("found Value : ", a)
                break
        else:
            print("not found")

    def traverse(self):
        a = []
        l = self.items
        for i in l:
            a.append(i)
        print(a)
```

Ln: 47 Col: 9

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 1.py - D:/DS/DS practi/Practical 1.py (3.7.0)

File Edit Format Run Options Window Help

```
#shorting
    def shoting_element(self):
        #bubble shotting
        nums=self.items
        def sort(nums):
            for i in range(len(nums) - 1, 0, -1):
                for j in range(i):
                    if nums[j] > nums[j + 1]:
                        temp = nums[j]
                        nums[j] = nums[j + 1]
                        nums[j + 1] = temp

        sort(nums)
        print(nums)
    def reverse(self):
        l=self.items
        print(l[::-1])
#class is made to merge two array
class merge1(Stack):
    #inheritance is created to merfe two array
    def __init__(self):
        Stack.__init__(self)
        self.items1 = array('i',[4,3,2,1,6])

    def merge(self):
        l = self.items
        l1=self.items1
        a=(l+l1)
        print(a)

s = Stack()
# Inserting the values
s.end(5)
s.end(6)
s.end(7)
s.start(-1)
s.start(-2)
print("search the specific value : ")
s.search(-2)

print("Display the values one by one :")
s.display()
print("peek (End  Value) :", s.peek())
print("treverse the values : ")
s.traverse()
```

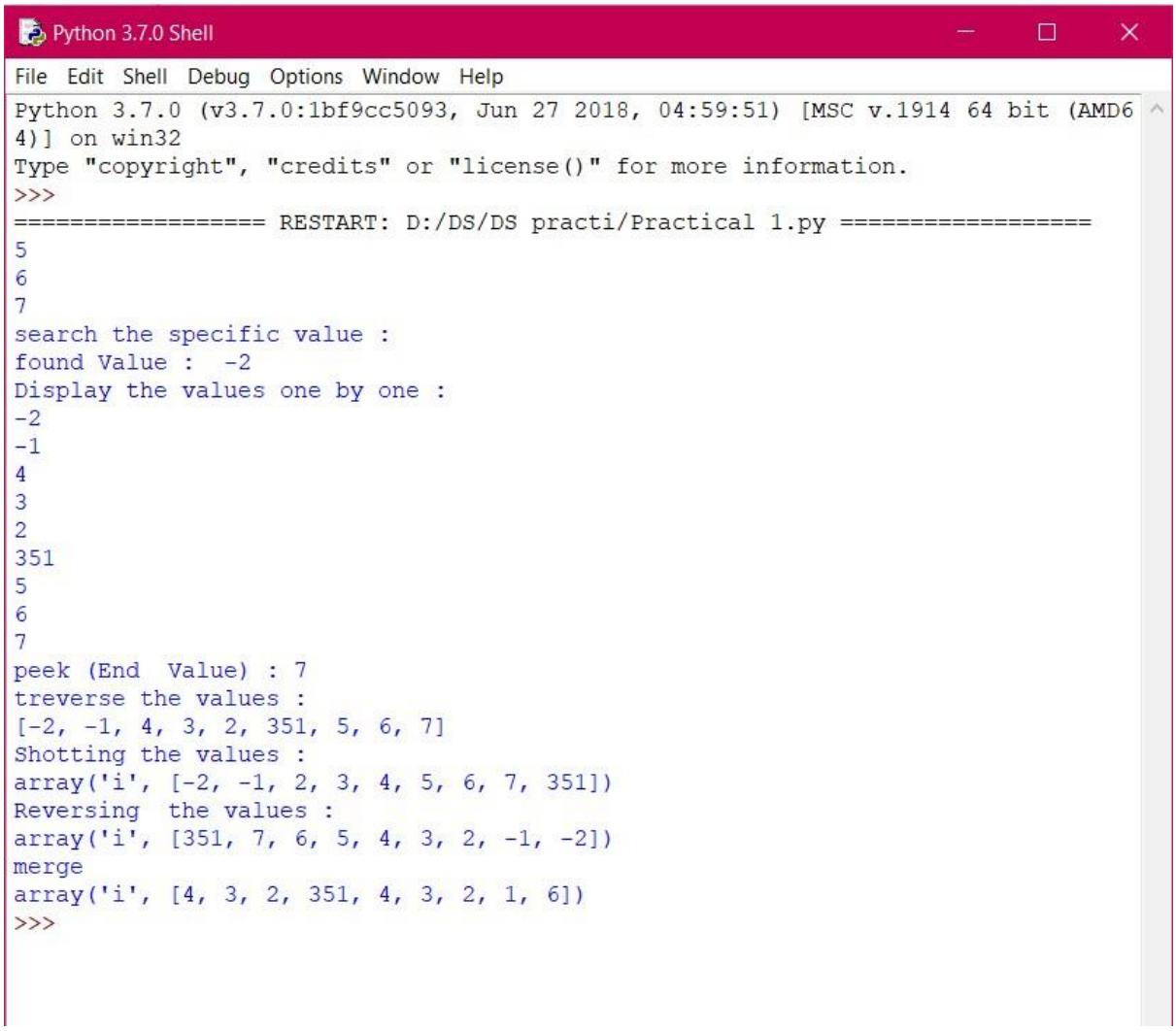
Ln: 91 Col: 12

```
#Shotting element
print("Shotting the values : ")
s.shoting_element()

print("Reversing  the values : ")
s.reverse()

s1=merge1()
print("merge")
s1.merge()
```

Ln: 91 Col: 12

Aniket Prajapati
SYCS –351
DS JOURNAL

Output: -



Aniket Prajapati
SYCS –351
DS JOURNAL
B)

Aim: -

Write a program to perform the Matrix addition, Multiplication and Transpose Operation.

Theory: -

Matrix is a special case of two-dimensional array where each data element is of strictly same size. So, every matrix is also a two-dimensional array but not vice versa. Matrices are very important data structures for many mathematical and scientific calculations. As we have already discussed two-dimensional array data structure in matrices.

In Python, we can implement a matrix as nested list (list inside a list).

We can treat each element as a row of the matrix. For example, X = [[1, 2], [4, 5], [3, 6]] would represent a 3x2 matrix. The first row can be selected as X[0]. And, the element in first row, first column can be selected as X[0][0]. Multiplication of two matrices X and Y is defined only if the number of columns in X is equal to the number of rows Y. If X is a n x m matrix and Y is a m x l matrix then, XY is defined and has the dimension n x l (but YX is not defined). Here are a couple of ways to implement matrix multiplication in Python.

Code:-

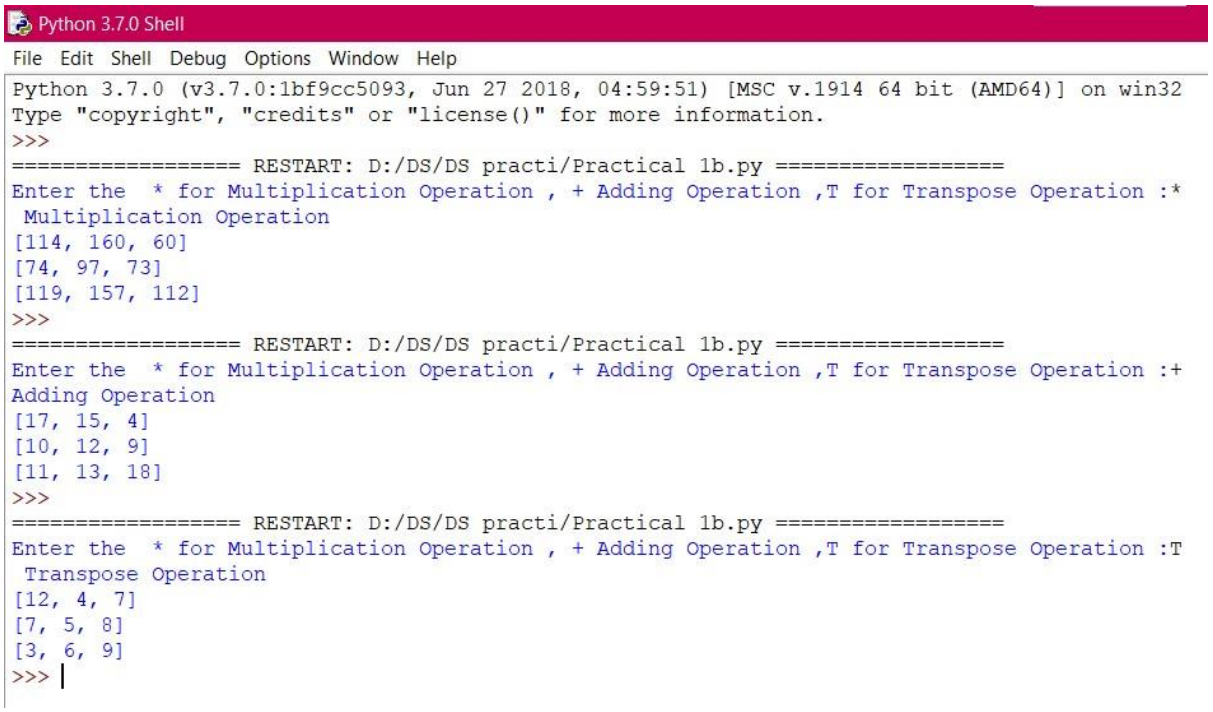
Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 1b.py - D:/DS/DS practi/Practical 1b.py (3.7.0)

File Edit Format Run Options Window Help

```
# Name  :Aniket Prajapati
# Rollno:351
# Class :SYCS
a=str(input("Enter the  * for Multiplication Operation , + Adding Operation ,T for Transpose Operation :"))
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]
if a== '+' :
    print("Adding Operation")
    def Adding_Matrix(X,Y):
        for i in range(len(X)):
   # iterate through columns
            for j in range(len(X[0])):
                result[i][j] = X[i][j] + Y[i][j]

        for r in result:
            print(r)

    Adding_Matrix(X,Y)
elif a == '*':
    print(" Multiplication Operation")
    for i in range(len(X)):
   # iterate through columns of Y
        for j in range(len(Y[0])):
       # iterate through rows of Y
            for k in range(len(Y)):
                result[i][j] += X[i][k] * Y[k][j]

    for r in result:
        print(r)

elif a == "t" or a == "T":
    print(" Transpose Operation")
    for i in range(len(X)):
   # iterate through columns
        for j in range(len(X[0])):
            result[j][i] = X[i][j]

    for r in result:
        print(r)

else:
    print("Enter valid input:")
```

Aniket Prajapati
SYCS –351
DS JOURNAL
Output: -



Aniket Prajapati
SYCS –351
DS JOURNAL

PRACTICAL NO -2

Aim: -

Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.

Theory: -

A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as discussed in the previous chapter. We have already seen how we create a node class and how to traverse the elements of a node. In this chapter we are going to study the types of linked lists known as singly linked lists. In this type of data structure there is only one link between any two data elements. We create such a list and create additional methods to insert, update and remove elements from the list.

Code: -

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 2.py - D:/DS/DS practi/Practical 2.py (3.7.0)

File Edit Format Run Options Window Help

```
# Name  :Aniket Prajapati
# Rollno:351
# Class :SYCS
class Stack():
    def __init__(self):
        self.items = ['4','3','2','1','Aniket']

    def end(self, item):
        self.items.append(item)
        print(item)

    def peek(self):
        if self.items:
            return self.items[-1]
        else:
            return None

    def size(self):
        if self.items:
            return len(self.items)
        else:
            return None

    def display(self):
        for i in self.items:
            print(i)

    def start(self, i):
        self.items.insert(0, i)

    def search(self, a):
        l = self.items
        for i in l:
            if i == a:
                print("found Value : ", a)
                break
        else:
            print("not found")
```

Ln: 40 Col: 23

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 2.py - D:/DS/DS practi/Practical 2.py (3.7.0)

File Edit Format Run Options Window Help

```
    def traverse(self):
        a = []
        l = self.items
        for i in l:
            a.append(i)
        print(a)
    def shoting_element(self):
        #bubble shotting
        nums=self.items
        def sort(nums):
            for i in range(len(nums) - 1, 0, -1):
                for j in range(i):
                    if nums[j] > nums[j + 1]:
                        temp = nums[j]
                        nums[j] = nums[j + 1]
                        nums[j + 1] = temp

        sort(nums)
        print(nums)
    #reverse
    def reverse(self):
        l=self.items
        print(l[::-1])

    def remove_value_from_particular_index(self,a):
        l=self.items
        l.pop(a)
        print(l)

class merge1(Stack):
    #inheritance
    def __init__(self):
        Stack.__init__(self)
        self.items1 = ['4','3','2','1','6']

    def merge(self):
        l = self.items
        l1=self.items1
        a=(l+l1)
        a.sort()
        print(a)
```

Ln: 80 Col: 16

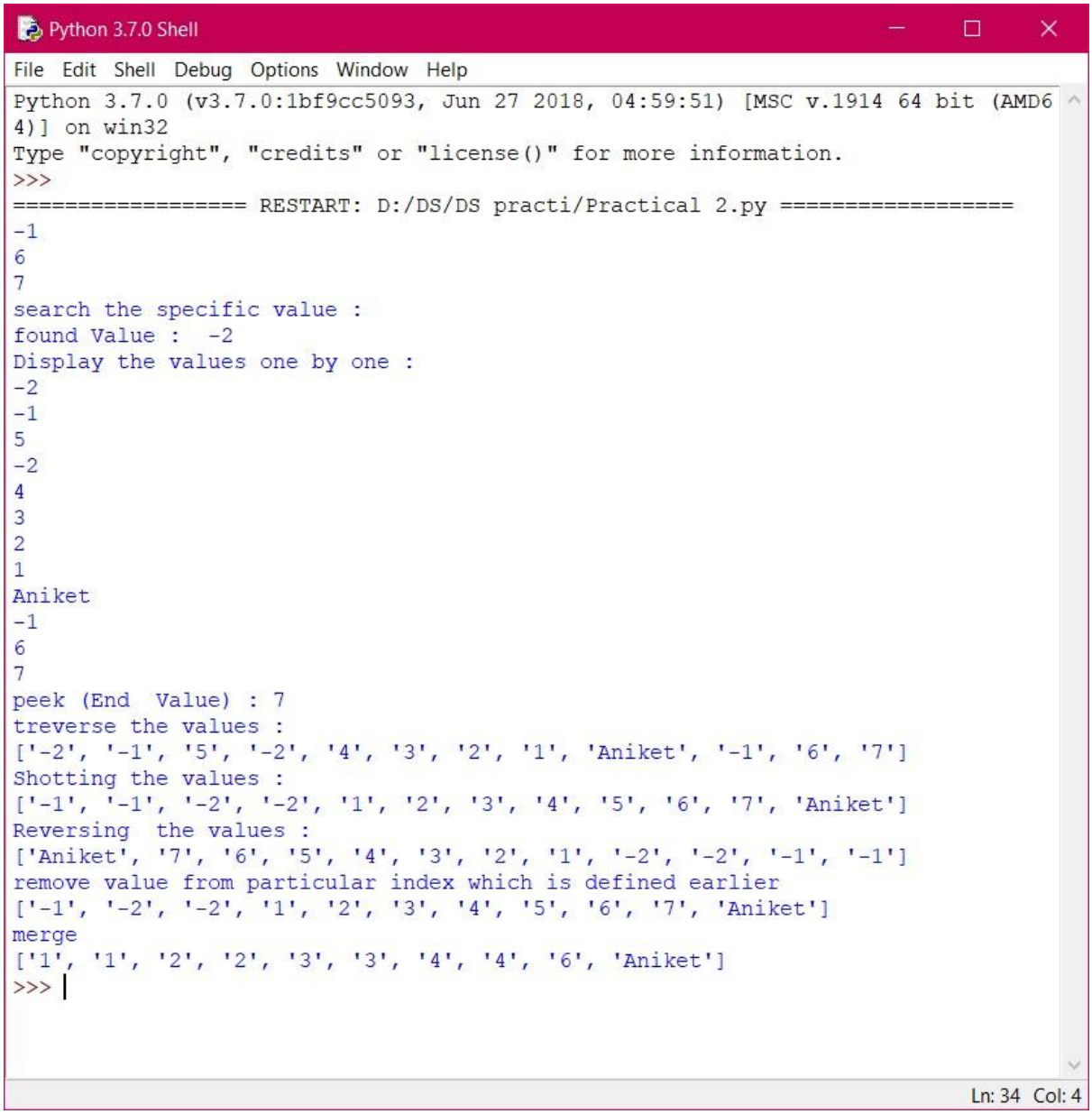
Aniket Prajapati
SYCS –351
DS JOURNAL

```
s = Stack()
# Inserting the values
s.end('-1')
s.start('-2')
s.start('5')
s.end('6')
s.end('7')
s.start('-1')
s.start('-2')
print("search the specific value : ")
s.search('-2')

print("Display the values one by one :")
s.display()
print("peek (End  Value) :", s.peek())
print("treverse the values : ")
s.traverse()
#Shotting element
print("Shotting the values : ")
s.shoting_element()
#reversing the list
print("Reversing  the values : ")
s.reverse()

print("remove value from particular index which is defined earlier")
s.remove_value_from_particular_index(0)

s1=merge1()
print("merge")
s1.merge()
```

Ln: 80 Col: 16

Output: -

Aniket Prajapati
SYCS –351
DS JOURNAL



Aniket Prajapati
SYCS –351
DS JOURNAL

PRACTICAL NO-3

Aim: -

Implement the following for Stack.

Theory: -

Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array**.**

Element– Each item stored in an array is called an element.

Index – Each location of an element in an array has a numerical index, which is used to identify the element**.**

Code: -

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 3a.py - D:\DS\DS practi\Practical 3a.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#351
#SYCS
from sys import maxsize


def createStack():
        stack = []
        return stack

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


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


def pop(stack):
        if (isEmpty(stack)):
                return str(-maxsize -1)

        return stack.pop()


def peek(stack):
        if (isEmpty(stack)):
                return str(-maxsize -1)
        return stack[len(stack) - 1]


stack = createStack()
push(stack, str(10))
push(stack, str(20))
push(stack, str(30))
print(pop(stack) + " popped from stack")
```

Ln: 38 Col: 0

Output: -

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/DS/DS practi/Practical 3a.py ==========
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
>>>
```

Aniket Prajapati
SYCS –351
DS JOURNAL

Aim: -

B) Implement tower of Hanoi

Theory: -

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules: -

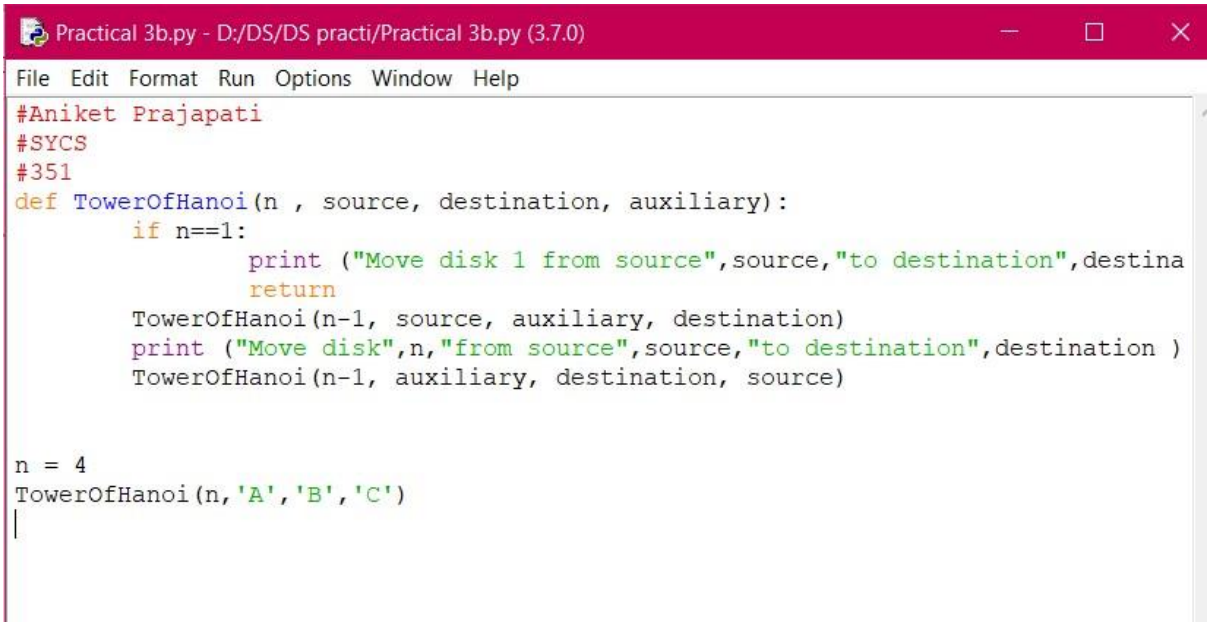
1) Only one disk can be moved at a time.

2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.

i.e., a disk can only be moved if it is the uppermost disk on a stack.

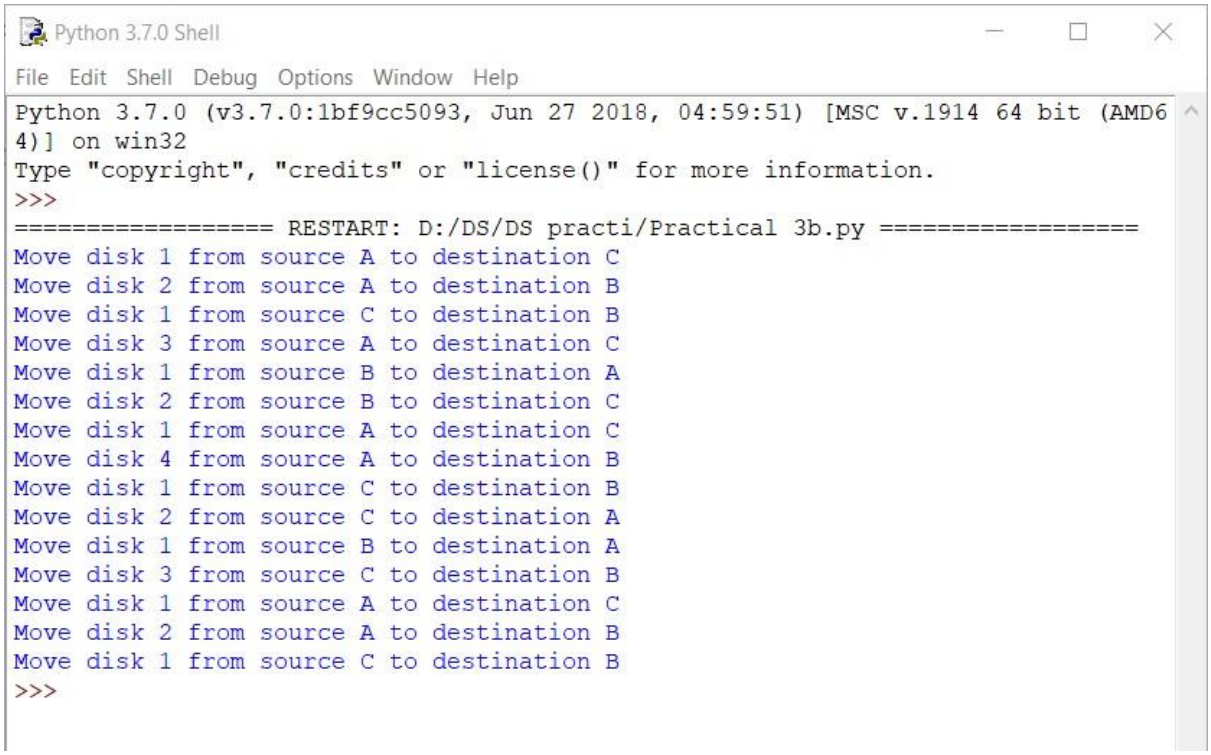
3) No disk may be placed on top of a smaller disk.

Code: -



Output: -

Aniket Prajapati
SYCS –351
DS JOURNAL



Aniket Prajapati
SYCS –351
DS JOURNAL

Aim: -

WAP to scan a polynomial using linked list and add two polynomials.

Theory: -

A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as discussed in the previous chapter. We have already seen how we create a node class and how to traverse the elements of a node. In this chapter we are going to study the types of linked lists known as singly linked lists. In this type of data structure there is only one link between any two data elements. We create such a list and create additional methods to insert, update and remove elements from the list.

Code: -

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 3c.py - D:/DS/DS practi/Practical 3c.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#SYCS
#351
def add(A, B, m, n):

        size = max(m, n);
        sum = [0 for i in range(size)]



        for i in range(0, m, 1):
                sum[i] = A[i]


        for i in range(n):
                sum[i] += B[i]

        return sum


def printPoly(poly, n):
        for i in range(n):
                print(poly[i], end = "")
                if (i != 0):
                        print("x^", i, end = "")
                if (i != n - 1):
                        print(" + ", end = "")


if __name__ == '__main__':


        A = [5, 0, 10, 6]


        B = [1, 2, 4]
        m = len(A)
        n = len(B)
```

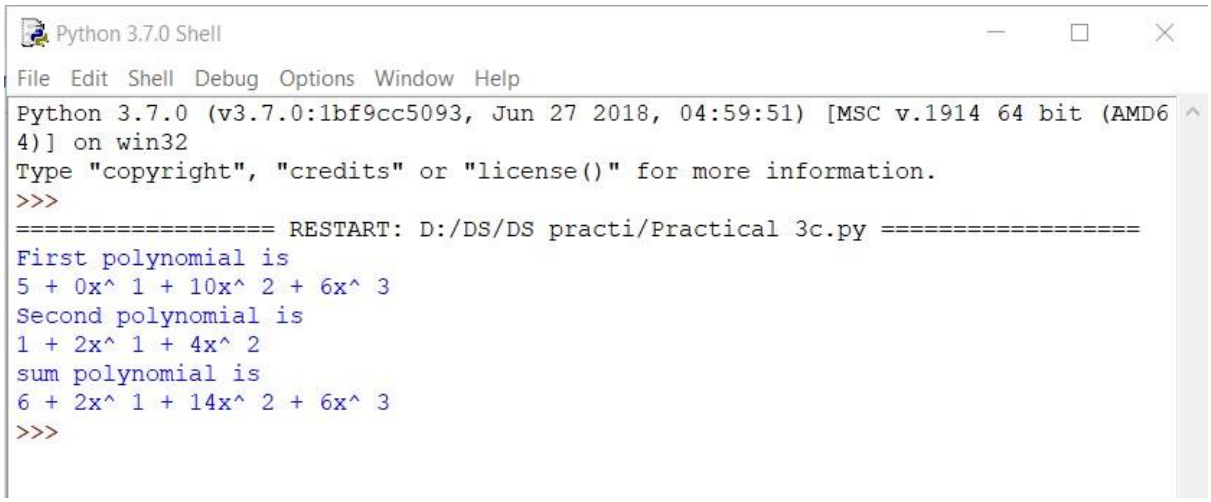
Ln: 38 Col: 12

```
        print("First polynomial is")
        printPoly(A, m)
        print("\n", end = "")
        print("Second polynomial is")
        printPoly(B, n)
        print("\n", end = "")
        sum = add(A, B, m, n)
        size = max(m, n)

        print("sum polynomial is")
        printPoly(sum, size)
```

Ln: 50 Col: 22

Aniket Prajapati
SYCS –351
DS JOURNAL

Output: -



Aniket Prajapati
SYCS –351
DS JOURNAL

Aim: -

D. WAP to calculate factorial and to compute the factors of a given no. (i) using recursion,

(i) using recursion

(ii) using iteration.

Code: -

Practical p1 3d.py - D:/DS/DS practi/Practical p1 3d.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#SYCS
#351
def recur_factorial(n):
   if n == 1:
       return n
   else:
       return n*recur_factorial(n-1)

num = int(input("Enter a number: "))

if num < 0:
   print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   print("The factorial of",num,"is",recur_factorial(num))
```

Output: -

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: D:/DS/DS practi/Practical p1 3d.py ================
Enter a number: 5
The factorial of 5 is 120
>>>
```

ii)USING ITERATION

Aniket Prajapati
SYCS –351
DS JOURNAL

CODE: -

Practical 3d.py - D:/DS/DS practi/Practical 3d.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#SYCS
#351
def fact(number):

    fact = 1

    for number in range(number, 1,-1):

        fact = fact * number
    return fact

number = int(input("Enter The Number : "))

factorial = fact(number)
print("Factorial is "+str(factorial))
```

OUTPUT: -

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/DS/DS practi/Practical 3d.py ==========
Enter The Number : 5
Factorial is 120
>>>
```

Aniket Prajapati
SYCS –351
DS JOURNAL

PRACTICAL NO-4

Aim: -

Perform Queues operations using Circular Array implementation.

Theory:

A Circular Queue is a queue data structure but circular in shape, therefore after the last position, the next place in the queue is the first position.

We recommend you to first go through the Linear Queue tutorial before Circular queue, as we will be extending the same implementation.

In case of Linear queue, we did not have the head and tail pointers because we used python List for implementing it. But in case of a circular queue, as the size of the queue is fixed, hence we will set a maxSize for our list used for queue implementation.

Code: -

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 4.py - D:/DS/DS practi/Practical 4.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#SYCS
#351
class CircularQueue:


  def __init__(self, maxSize):
    self.queue = list()
    self.maxSize = maxSize
    self.head = 0
    self.tail = 0


  def enqueue(self, data):
    if self.size() == (self.maxSize - 1):
      return("Queue is full!")
    else:
        self.queue.append(data)

        self.tail = (self.tail+1) % self.maxSize
        return True


  def dequeue(self):
    if self.size() == 0:
      return("Queue is empty!")
    else:
        data = self.queue[self.head]

        self.head = (self.head+1) % self.maxSize
        return data


  def size(self):
    if self.tail >= self.head:
      qSize = self.tail - self.head
    else:
      qSize = self.maxSize - (self.head - self.tail)
    return qSize
```

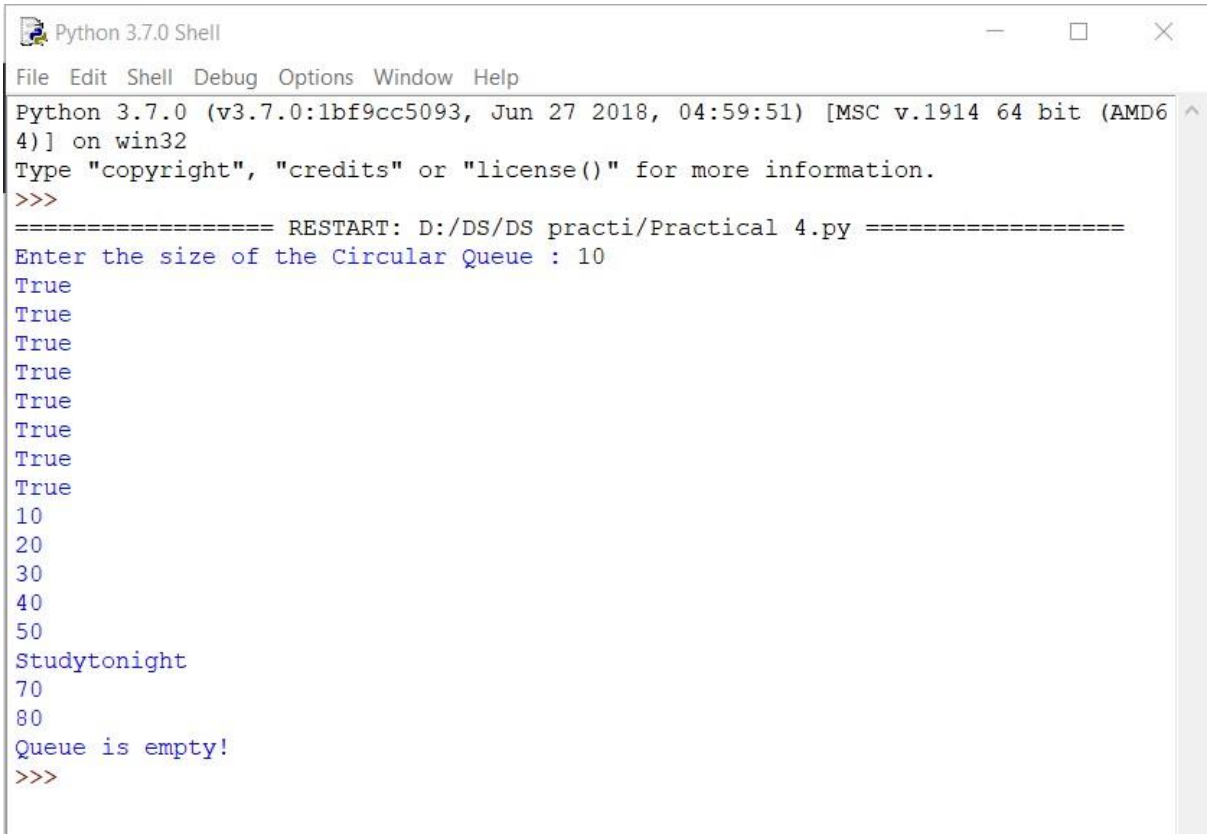
Ln: 40 Col: 0

Aniket Prajapati
SYCS –351
DS JOURNAL

```
size = input("Enter the size of the Circular Queue : ")
q = CircularQueue(int(size))


print(q.enqueue(10))
print(q.enqueue(20))
print(q.enqueue(30))
print(q.enqueue(40))
print(q.enqueue(50))
print(q.enqueue('Studytonight'))
print(q.enqueue(70))
print(q.enqueue(80))
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
```

Ln: 62 Col: 18

Output: -



Aniket Prajapati
SYCS –351
DS JOURNAL

PRACTICAL NO-5

Aim: -

Write a program to search an element from a list. Give user the option to perform Linear or Binary search.

Theory: -

Searching is a very basic necessity when you store data in different data structures. The simplest approach is to go across every element in the data structure and match it with the value you are searching for. This is known as Linear search. It is inefficient and rarely used, but creating a program for it gives an idea about how we can implement some advanced search algorithms.

Linear Search: -

In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data structure.

Binary Search: -

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Code: -

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 5.py - D:/DS/DS practi/Practical 5.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#351
#SYCS
a = str(input("Enter the string  l for Linear Search and b For Binary Search: "))
pos = -1
list = [0,1,2,3,4,5,6,7,8,45,63]
if a == 'b':
    def search(list,n):
        l = 0
        u = len(list)-1
        while l <=u:
            mid = (l+u)//2

            if list[mid] == n:
                globals()['pos']= mid
                return True
            else:
                if list[mid]<n:
                    l = mid+1
                else:
                    u = mid-1
        return False
    #list = [0,1,2,3,4,5,6,7,8,45,63]
    list.sort()
    n= int(input("Enter the numbers for binary  search : "))
    if search(list, n):
        print("Number Found  : ")
    else:
        print("Not Found : ")
elif a == 'l':
    #pos = -1
    def search(list ,n):
        i = 0

        while i < len(list):
            if list[i] == n:
                return True
            i = i+1
```

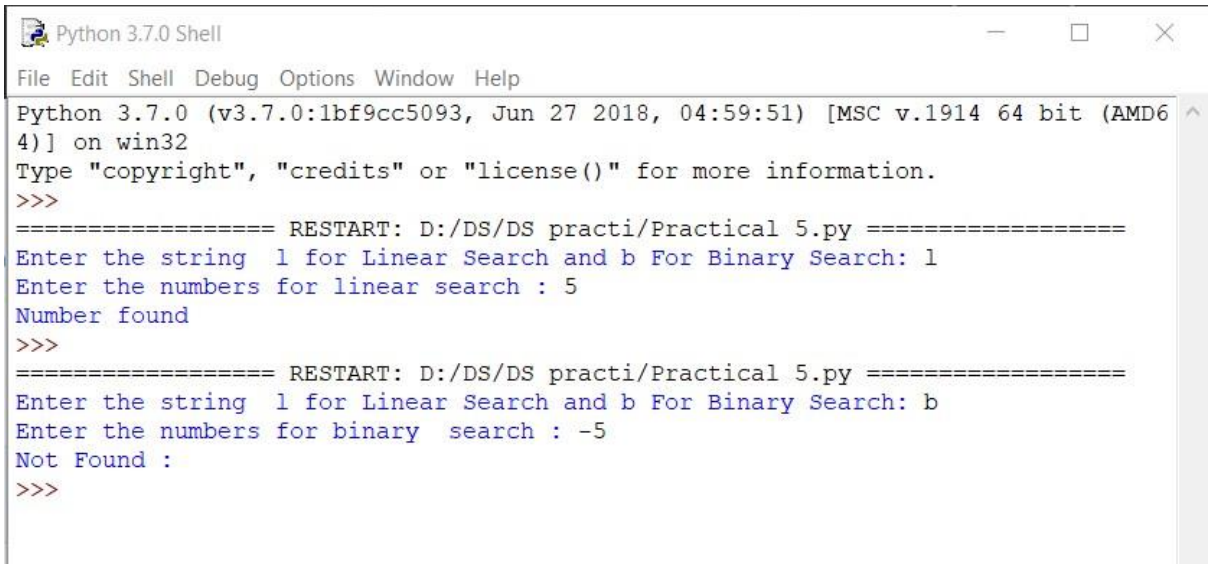
Ln: 38 Col: 19

```
        return False

    list.sort()
    n= int(input("Enter the numbers for linear search : "))
    if search(list ,n):
        print("Number found ")
    else:
        print("not found")

else:
    print("enter valid input")
```

Ln: 38 Col: 19

Aniket Prajapati
SYCS –351
DS JOURNAL

Output: -



Aniket Prajapati
SYCS –351
DS JOURNAL

PRACTICAL NO-6

Aim: -

WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

Theory:

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Below we see five such implementations of sorting in python**.**

Bubble Sort

It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

Insertion Sort

Insertion sort involves finding the right place for a given element in a sorted list. So in beginning we compare the first two elements and sort them by comparing them. Then we pick the third element and find its proper position among the previous two sorted elements. This way we gradually go on adding more elements to the already sorted list by putting them in their proper position.

Selection Sort

In selection sort we start by finding the minimum value in a given list and move it to a sorted list. Then we repeat the process for each of the remaining elements in the unsorted list. The next element entering the sorted list is compared with the existing elements and placed at its correct position. So at the end all the elements from the unsorted list are sorted**.**

Code: -

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 6.py - D:/DS/DS practi/Practical 6.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#351
#SYCS

nums = [5,4,4063,-1]
a = str(input("enter the string i for insertion sort , b for bubble sort , s for
if a=='i' or a =='I':

    def insertion_sort(nums):
        for i in range(1, len(nums)):
            j = i-1
            nxt_element = nums[i]


            while (nums[j] > nxt_element) and (j >= 0):
                nums[j+1] = nums[j]
                j=j-1
            nums[j+1] = nxt_element


    insertion_sort(nums)
    print(nums)
elif a == 'b' or a == 'B':

    def sort(nums):
        for i in range(len(nums)-1,0,-1):
            for j in range(i):
                if nums[j]>nums[j+1]:
                    temp = nums[j]
                    nums[j]=nums[j+1]
                    nums[j+1] = temp
    sort(nums)
    print(nums)
elif a == 's' or a =='S':
    def sort(nums):
        for i in range(len(nums)):
            minpos = i
            for j in range(i,len(nums)):
                if nums[j] < nums[minpos]:
                    minpos=j
```

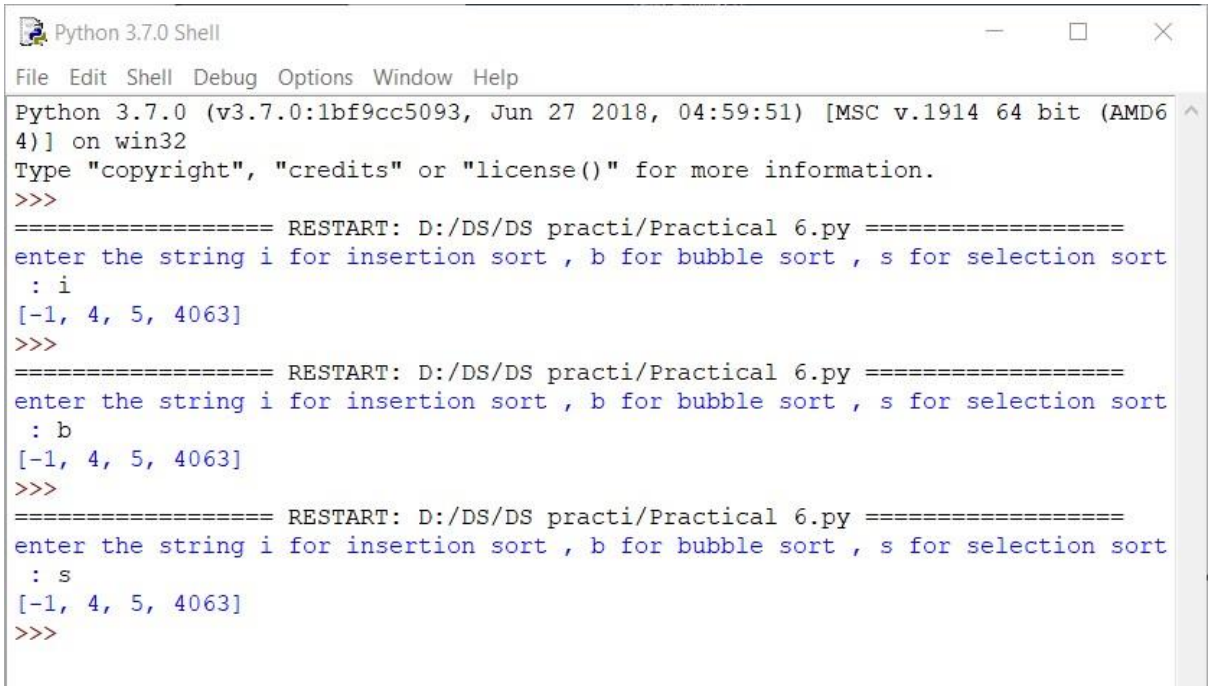
Ln: 17 Col: 0

```
            temp = nums[i]
            nums[i] = nums[minpos]
            nums[minpos] =temp


    sort(nums)
    print(nums)
else:
    print("Enter valid input")
```

Ln: 17 Col: 0

Aniket Prajapati
SYCS –351
DS JOURNAL

OUTPUT: -



Aniket Prajapati
SYCS –351
DS JOURNAL

PRACTICAL NO-7

Aim: - Implement the following for Hashing

A. Write a program to implement the collision technique

Hash tables are a type of data structure in which the address or the index value of the data element is generated from a hash function. That makes accessing the data faster as the index value behaves as a key for the data value. In other words Hash table stores key-value pairs but the key is generated through a hashing function.

So the search and insertion function of a data element becomes much faster as the key values themselves become the index of the array which stores the data.

In Python, the Dictionary data types represent the implementation of hash tables. The Keys in the dictionary satisfy the following requirements**.**

1.The keys of the dictionary are hash able i.e. the are generated by hashing function which generates unique result for each unique value supplied to the hash function.

2. The order of data elements in a dictionary is not fixed**.**

Code: -

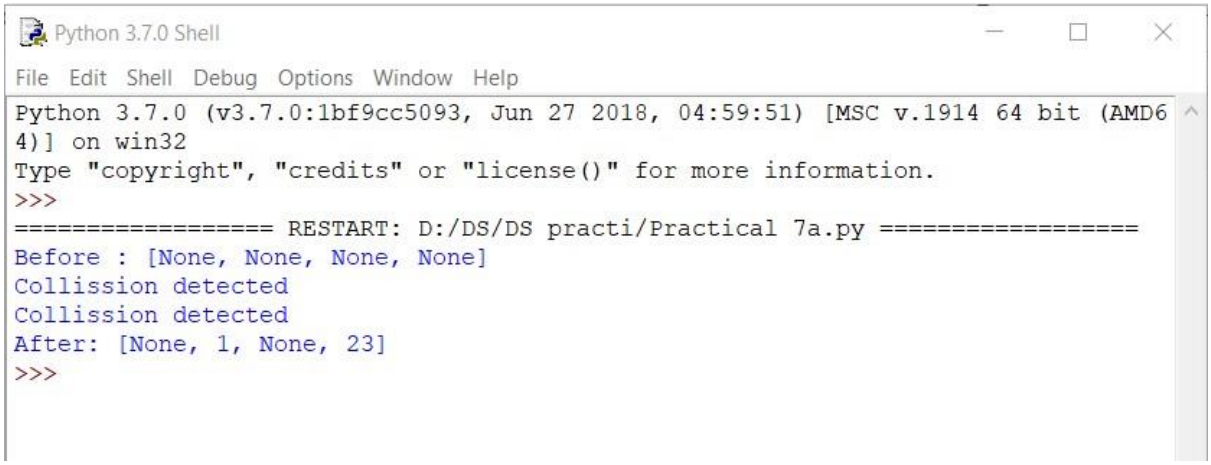
Practical 7a.py - D:/DS/DS practi/Practical 7a.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#351
#SYCS
class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys,lowerrange, higherrange)

    def get_key_value(self):
        return self.value

    def hashfunction(self,keys,lowerrange, higherrange):
        if lowerrange == 0 and higherrange > 0:
            return keys%(higherrange)

if __name__ == '__main__':
    list_of_keys = [23,43,1,87]
    list_of_list_index = [None,None,None,None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:

        list_index = Hash(value,0,len(list_of_keys)).get_key_value()
        if list_of_list_index[list_index]:
            print("Collission detected")
        else:
            list_of_list_index[list_index] = value

    print("After: " + str(list_of_list_index))
```

Aniket Prajapati
SYCS –351
DS JOURNAL

Output: -



Aniket Prajapati
SYCS –351
DS JOURNAL

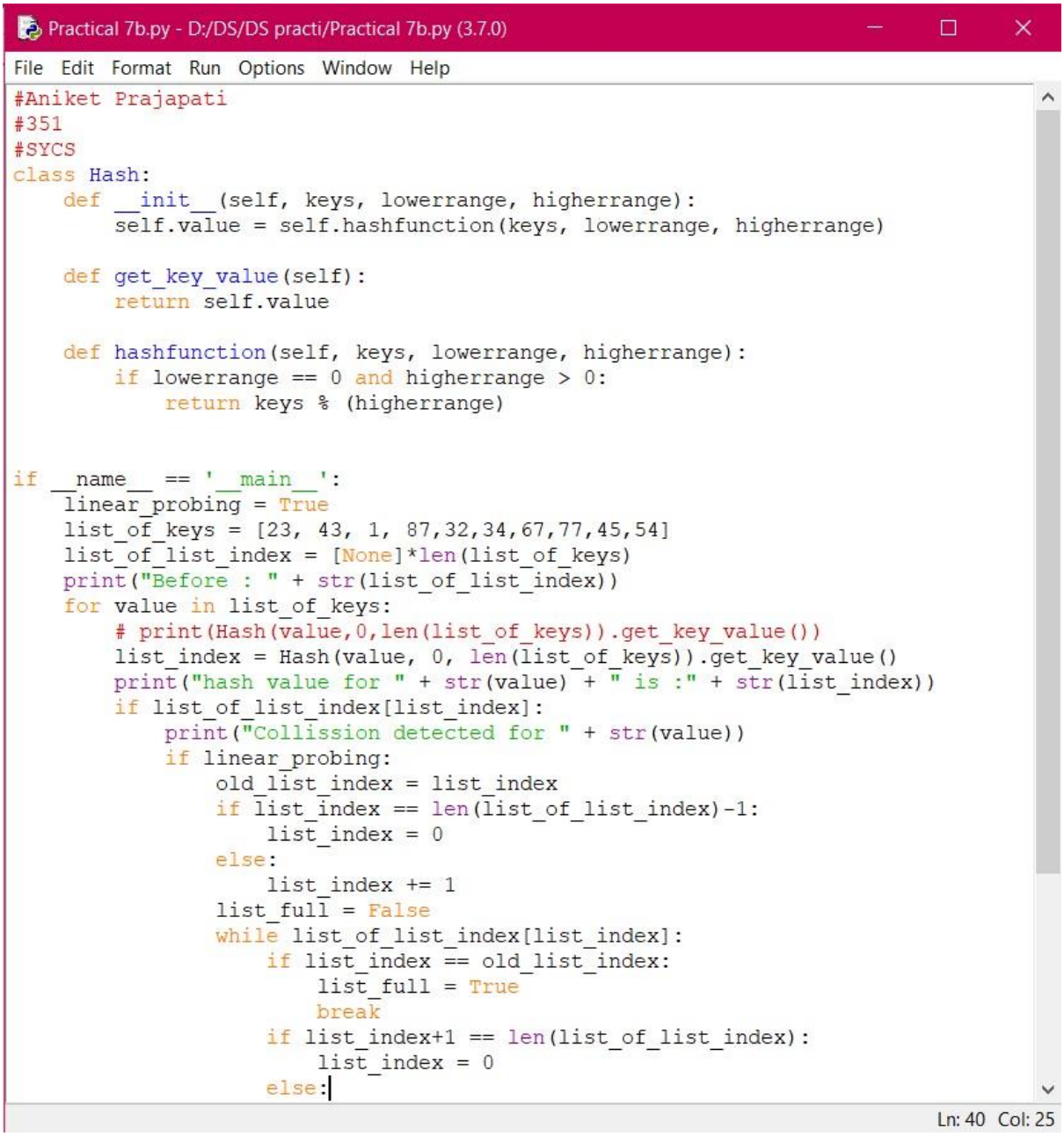
Aim: -

B. Write a program to implement the concept of linear probing

Theory: -

Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of table must be greater than or equal to total number of keys (Note that we can increase table size by copying old data if needed).

Code: -



```
                    list_index += 1
                if list_full:
                    print("List was full . Could not save")
                else:
                    list_of_list_index[list_index] = value

        else:
            list_of_list_index[list_index] = value

    print("After: " + str(list_of_list_index))
```

Ln: 40 Col: 25

Output: -

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: D:/DS/DS practi/Practical 7b.py ==================
Before : [None, None, None, None, None, None, None, None, None, None]
hash value for 23 is :3
hash value for 43 is :3
Collission detected for 43
hash value for 1 is :1
hash value for 87 is :7
hash value for 32 is :2
hash value for 34 is :4
Collission detected for 34
hash value for 67 is :7
Collission detected for 67
hash value for 77 is :7
Collission detected for 77
hash value for 45 is :5
Collission detected for 45
hash value for 54 is :4
Collission detected for 54
After: [54, 1, 32, 23, 43, 34, 45, 87, 67, 77]
>>>
```

Aniket Prajapati
SYCS –351
DS JOURNAL

PRACTICAL NO-8

Aim: -

Write a program for inorder, postorder and preorder traversal of tree.

Theory:

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

Depth First Traversals:

(a) Inorder (Left, Root, Right)

(b) Preorder (Root, Left, Right)

(c) Postorder (Left, Right, Root)

Code: -

Aniket Prajapati
SYCS –351
DS JOURNAL

Practical 8.py - D:/DS/DS practi/Practical 8.py (3.7.0)

File Edit Format Run Options Window Help

```
#Aniket Prajapati
#351
#SYCS
import random

random.seed(23)

class Node:
    def __init__(self, val):
        self.val = val
        self.leftChild = None
        self.rightChild = None


def insert(root, key):
    if root is None:
        return Node(key)

    else:
        if root.val == key:
            return root
        elif root.val < key:
            root.rightChild = insert(root.rightChild, key)
        else:
            root.leftChild = insert(root.leftChild, key)

    return root


def PrintInorder(root):
    if root:
        PrintInorder(root.leftChild)
        print(root.val, end=" ")
        PrintInorder(root.rightChild)


def printPreorder(root):
    if root:
        print(root.val, end=" ")
        printPreorder(root.leftChild)
        printPreorder(root.rightChild)
```

Ln: 41 Col: 38

Aniket Prajapati
SYCS –351
DS JOURNAL

```
def printPostorder(root):
    if root:
        printPostorder(root.leftChild)
        printPostorder(root.rightChild)
        print(root.val, end=" ")


tree = Node(20)
for i in range(10):
    insert(tree, random.randint(2, 100))


if __name__ == "__main__":
    print("inorder")
    PrintInorder(tree)
    print("\n")
    print("preorder")
    printPreorder(tree)
    print("\n")
    print("postorder")
    printPostorder(tree)
```

Ln: 41 Col: 38

OUTPUT: -