MALAD KANDIVALI EDUCATION SOCIETY'S

NAGINDAS KHANDWALA COLLEGE OF COMMERCE, ARTS & MANAGEMENT STUDIES & SHANTABEN NAGINDAS KHANDWALA COLLEGE OF SCIENCE

MALAD [W], MUMBAI – 64

AUTONOMOUS INSTITUTION

(Affiliated To University Of Mumbai)

Reaccredited 'A' Grade by NAAC | ISO 9001:2015 Certified

CERTIFICATE

Name: Ms. Lakshita . Narsian

Roll No: 328 Programme: BSc IT Semester: III

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **Data Structures (Course Code: 2032UISPR)** for the partial fulfilment of Third Semester of BSc IT during the academic year 2020-21.

The journal work is the original study work that has been duly approved in the year 2020-21 by the undersigned.

External Examiner

Mr. Gangashankar Singh
(Subject-In-Charge)

Date of Examination: (College Stamp)

Class: S.Y. B.Sc. IT Sem- III **Roll No: 328**

Subject: Data Structures

INDEX

Sr No	Date	Topic	Sign
1	04/09/2020	Implement the following for Array: 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. b) Write a program to perform the Matrix addition, Multiplication and Transpose Operation.	
2	11/09/2020	Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.	
3	18/09/2020	Implement the following for Stack: a) Perform Stack operations using Array implementation. b. b) Implement Tower of Hanoi. c) WAP to scan a polynomial using linked list and add two polynomials. d) WAP to calculate factorial and to compute the factors of a given no. (i) using recursion, (ii) using iteration	
4	25/09/2020	Perform Queues operations using Circular Array implementation.	
5	01/10/2020	Write a program to search an element from a list. Give user the option to perform Linear or Binary search.	
6	09/10/2020	WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.	
7	16/10/2020	Implement the following for Hashing: a) Write a program to implement the collision technique. b) Write a program to implement the concept of linear probing.	
8	23/10/2020	Write a program for inorder, postorder and preorder traversal of tree.	

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

PRACTICAL NO: 1

AIM –

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

THEORY –

One dimensional array contains elements only in one dimension. In other words, the shape of the numpy array should contain only one value in the tuple.
Numpy array() functions takes a list of elements as argument and returns a one-dimensional array.
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.
Linear Search: A Linear Search is the most basic type of searching algorithm. A Linear Search sequentially moves through your collection (or data structure) looking for a matching value. In other words, it looks down a list, one item at a time, without jumping.
Bubble Sort: Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst-case complexity are of O(n2) where n is the number of items.
Selection Sort: 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 the unsorted subarray is picked and moved to the sorted subarray.
Insertion Sort: Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.
Merge: First we have to copy all the elements of the first list into a new list. Using a for loop we can append every element of the second list in the new list.
Reverse: First we have to create a new list. Using a reversed for loop we can append all the elements of the original list into the new list in reverse order.

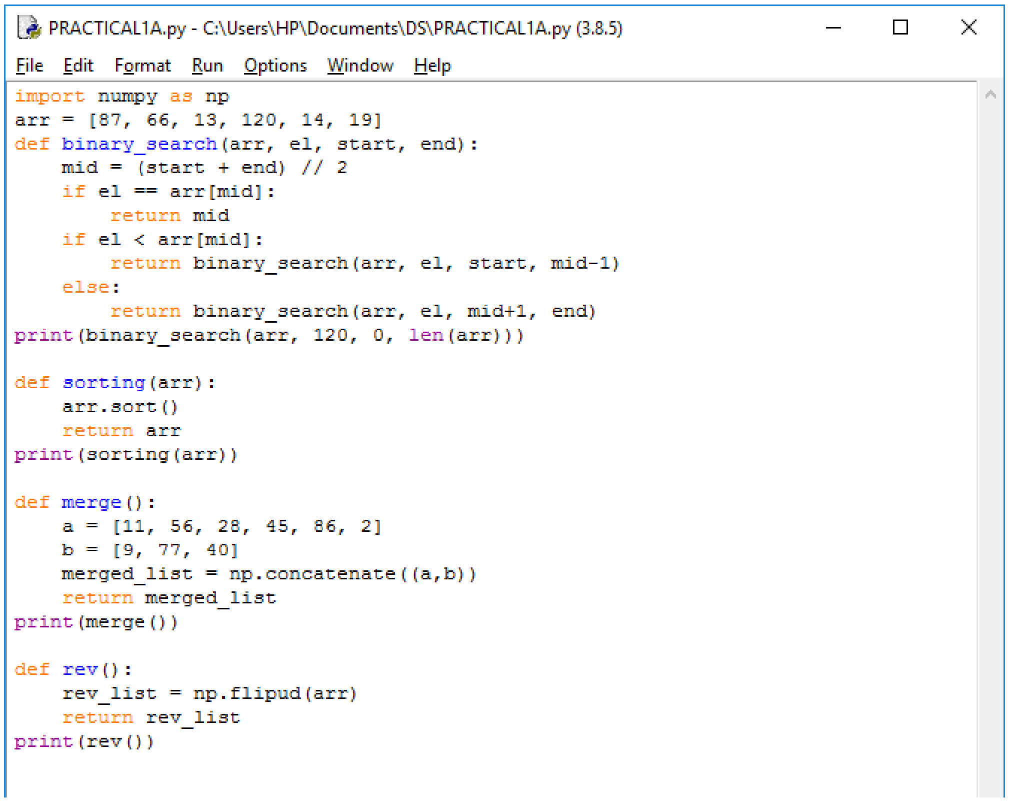
Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/Prcatical1A.py

CODE –

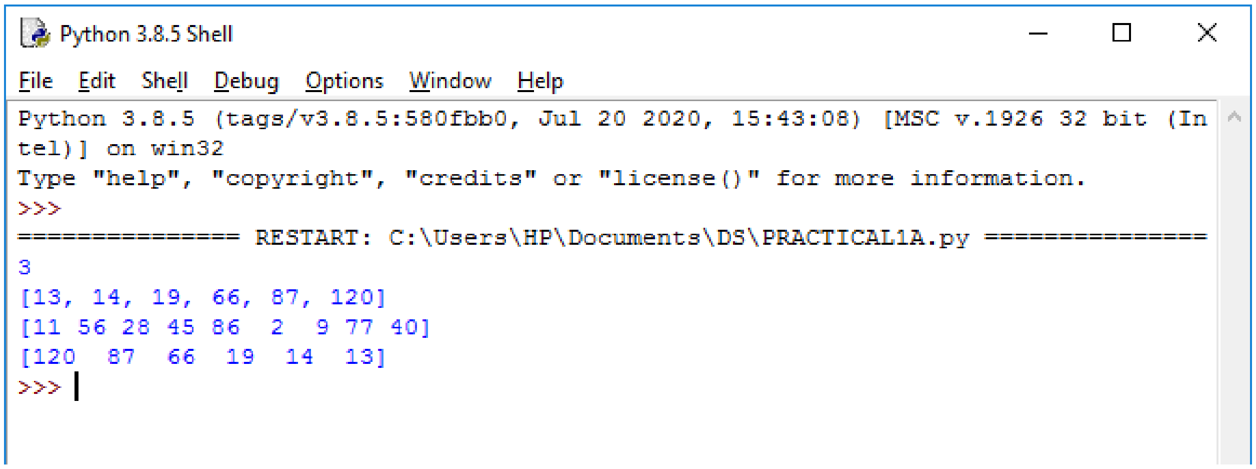
PRACTICAL 1 A:


PRACTICAL1A.py - C:\Users\HP\Documents\DS\PRACTICAL1A.py (3.8.5)
File Edit Format Run Options Window Help

```
import numpy as np
arr = [87, 66, 13, 120, 14, 19]
def binary_search(arr, el, start, end):
    mid = (start + end) // 2
    if el == arr[mid]:
        return mid
    if el < arr[mid]:
        return binary_search(arr, el, start, mid-1)
    else:
        return binary_search(arr, el, mid+1, end)
print(binary_search(arr, 120, 0, len(arr)))

def sorting(arr):
    arr.sort()
    return arr
print(sorting(arr))

def merge():
    a = [11, 56, 28, 45, 86, 2]
    b = [9, 77, 40]
    merged_list = np.concatenate((a,b))
    return merged_list
print(merge())

def rev():
    rev_list = np.flipud(arr)
    return rev_list
print(rev())
```

OUTPUT –

PRACTICAL 1 A:



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL1A.py ===============
3
[13, 14, 19, 66, 87, 120]
[11 56 28 45 86  2  9 77 40]
[120  87  66  19  14  13]
>>>
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

AIM -

1 B : Write a program to perform the Matrix addition, Multiplication and Transpose.

THEORY –

A Python matrix is a specialized two-dimensional rectangular array of data stored in rows and columns. The data in a matrix can be numbers, strings, expressions, symbols, etc. Matrix is one of the important data structures that can be used in mathematical and scientific calculations.
The python matrix makes use of arrays, and the same can be implemented.

- **Create a Python Matrix using the nested list data type**
- **Create Python Matrix using Arrays from Python Numpy package**

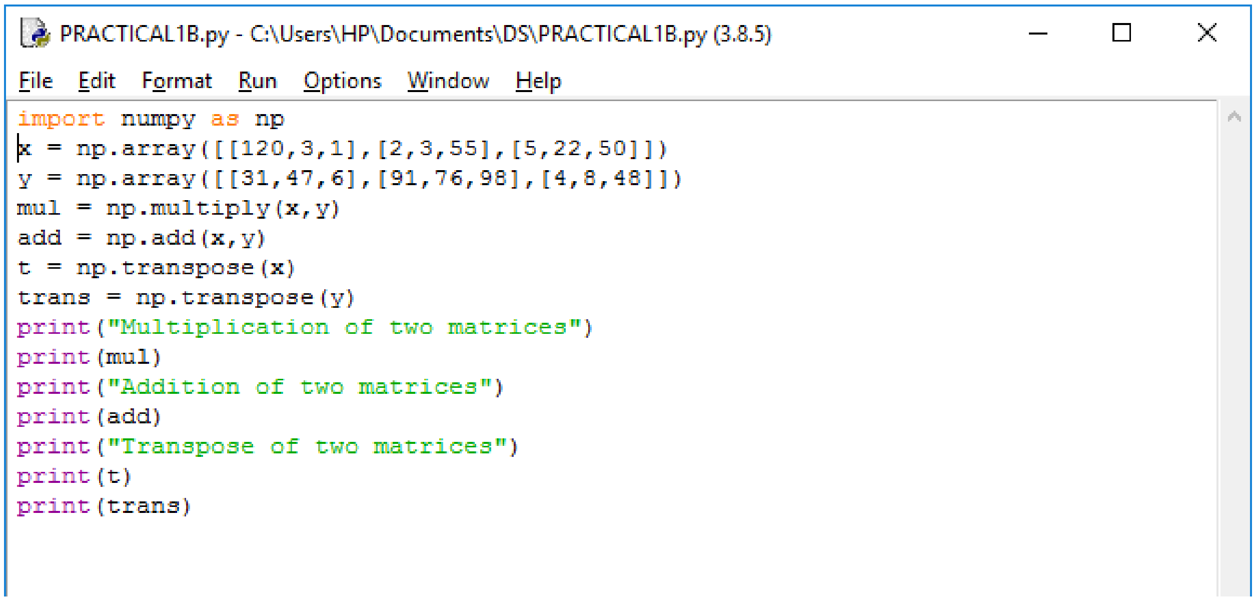
---Addition---
To add, the matrices will make use of a for-loop that will loop through both the matrices given.
---Multiplication---
To multiply the matrices, we can use the for-loop on both the matrices .
---Transpose---
Transpose of a matrix is the interchanging of rows and columns.

GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/Practical1B.py

CODE –

PRACTICAL 1 B:

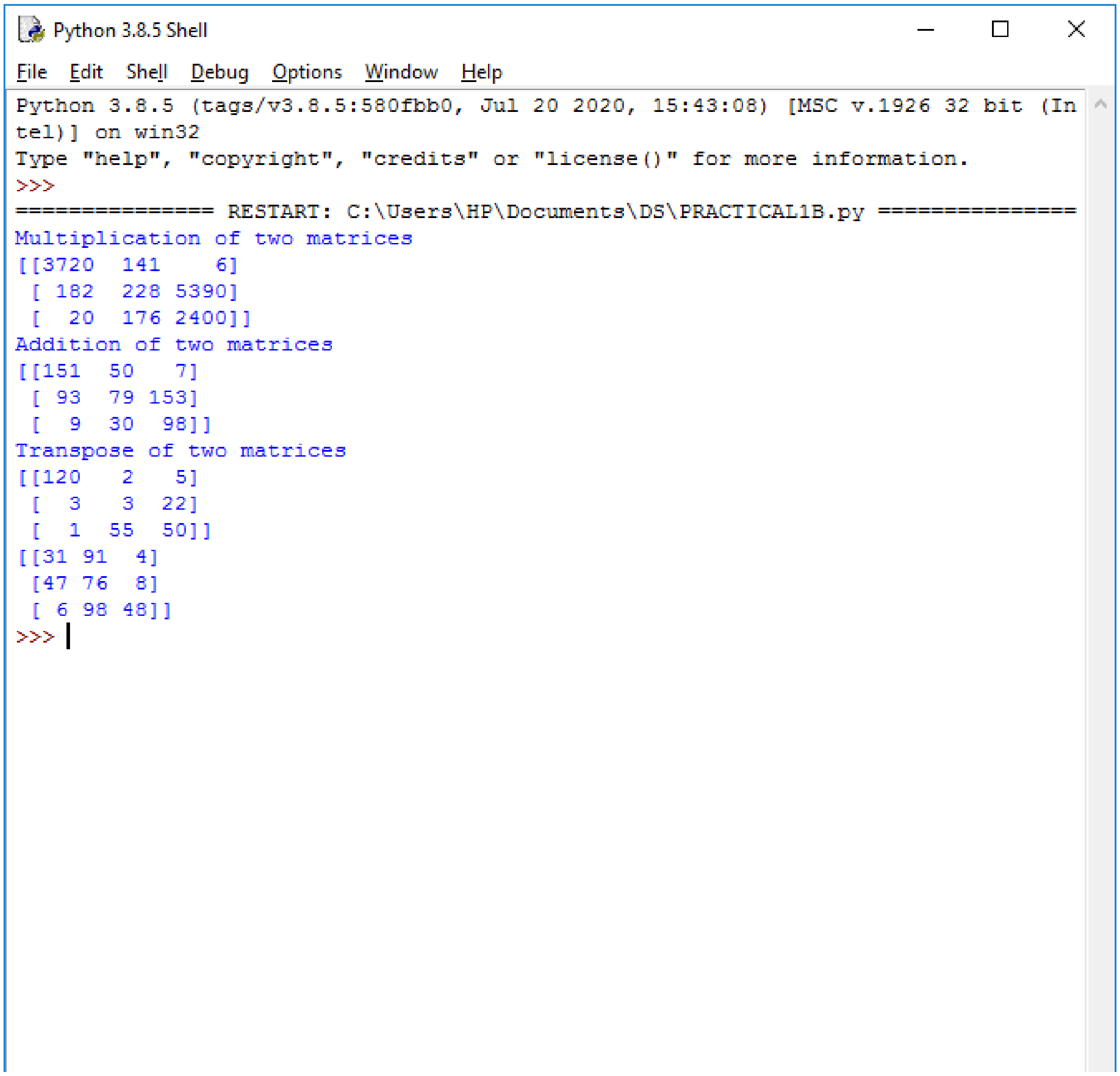

PRACTICAL1B.py - C:\Users\HP\Documents\DS\PRACTICAL1B.py (3.8.5)
File Edit Format Run Options Window Help

```
import numpy as np
x = np.array([[120,3,1],[2,3,55],[5,22,50]])
y = np.array([[31,47,6],[91,76,98],[4,8,48]])
mul = np.multiply(x,y)
add = np.add(x,y)
t = np.transpose(x)
trans = np.transpose(y)
print("Multiplication of two matrices")
print(mul)
print("Addition of two matrices")
print(add)
print("Transpose of two matrices")
print(t)
print(trans)
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

OUTPUT –

PRACTICAL 1 B:



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL1B.py ===============
Multiplication of two matrices
[[3720  141    6]
 [ 182  228 5390]
 [  20  176 2400]]
Addition of two matrices
[[151  50   7]
 [ 93  79 153]
 [  9  30  98]]
Transpose of two matrices
[[120   2   5]
 [  3   3  22]
 [  1  55  50]]
[[31 91  4]
 [47 76  8]
 [ 6 98 48]]
>>> 
```

Name: LAKSHITA. NARSIAN　　Class: SYIT　　Rollno: 328

PRACTICAL NO: 2

AIM –

Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate 2 linked lists.

THEORY –

A linked list is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence.
The principal benefit of a linked list over a conventional array is that the list elements can be easily inserted or removed without reallocation or reorganization of the entire structure.
Following are the various types of linked list :-

- **Simple Linked List - Item navigation is forward only.**
- **Doubly Linked List - Items can be navigated forward and backward.**
- **Circular Linked List - Last item contains link of the first element as next and the first element has a link to the last element as previous.**

Basic Operations:
Insertion, Deletion, Display, Search, Delete
<u>Insertion Operation</u>: Adding a new node in linked list is a more than one step activity. First, create a node using the same structure and find the location where it has to be inserted.

<u>Deletion Operation</u>: Deletion is also a more than one step process. First, locate the target node to be removed, by using searching algorithms.
Reverse Operation: This operation is a thorough one. We need to make the last node to be pointed by the head node and reverse the whole linked list

<u>Searching</u> is performed in order to find the location of a particular element in the list. Searching any element in the list needs traversing through the list and make the comparison of every element of the list with the specified element.
If the element is matched with any of the list element then the location of the element is returned from the function.

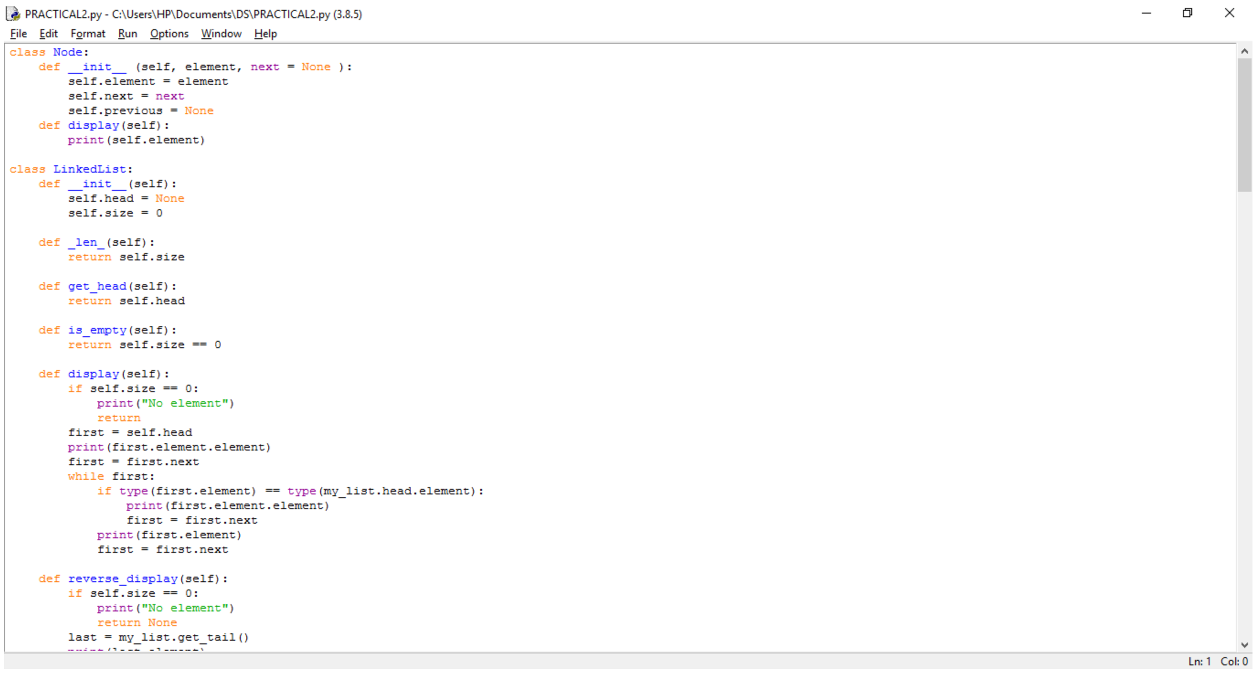
<u>Concatenate 2 linked lists</u> :
To concatenate means to join. To join two lists (say ‘a’ and ‘b’) are as follows:
Traverse over the linked list ‘a’ until the element next to the node is not NULL.
If the element next to the current element is NULL (a->next == NULL) then change the element next to it to ‘b’ (a->next = b).

GITHUB URL –

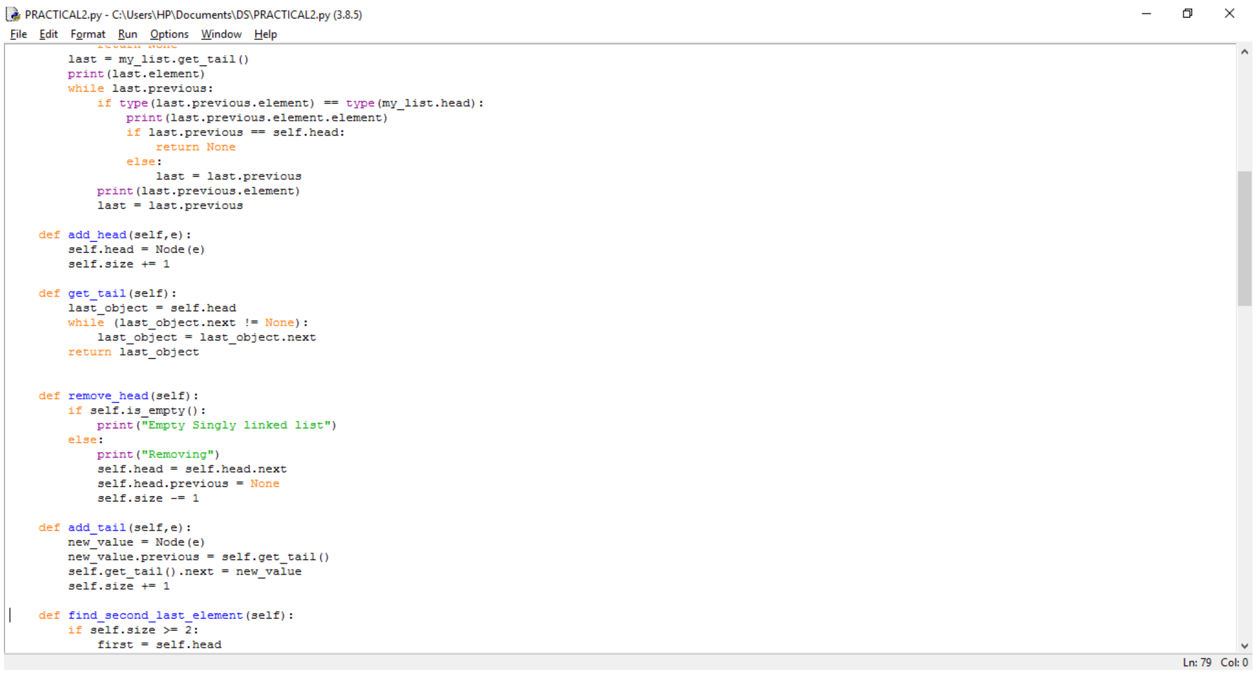
https://github.com/LakshitaNarsian/DS/blob/master/Practical2.py

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

CODE –

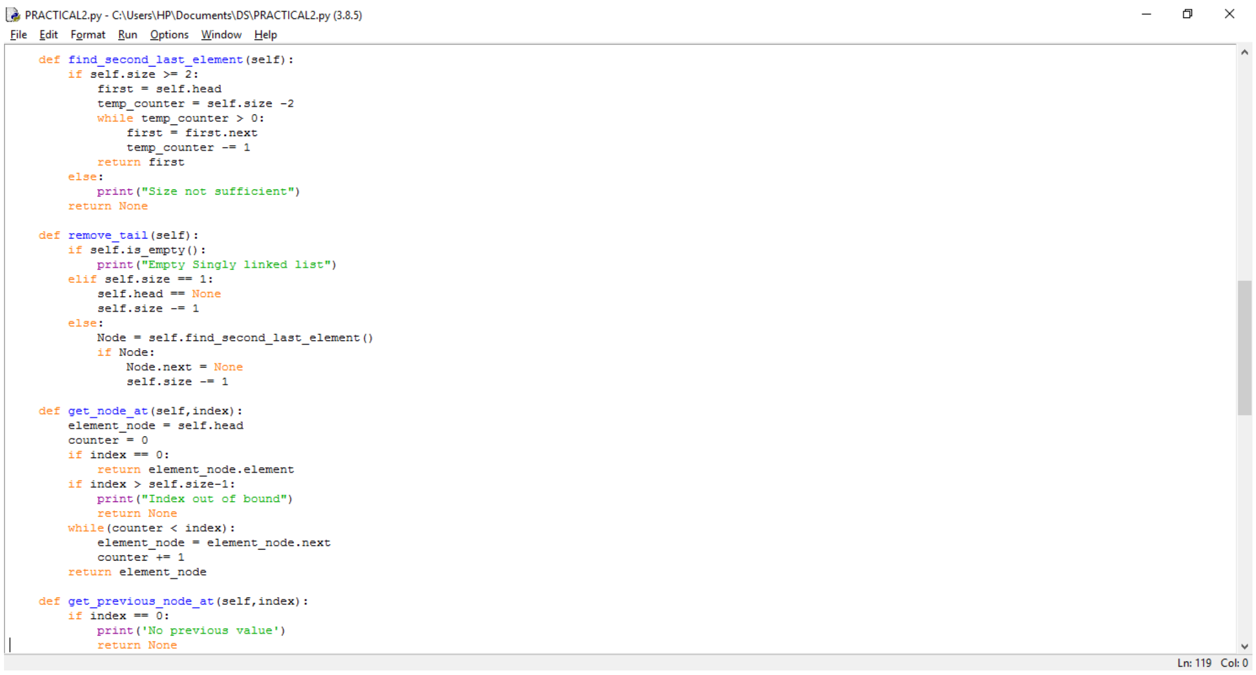

PRACTICAL2.py - C:\Users\HP\Documents\DS\PRACTICAL2.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```python
class Node:
    def __init__ (self, element, next = None ):
        self.element = element
        self.next = next
        self.previous = None
    def display(self):
        print(self.element)

class LinkedList:
    def __init__(self):
        self.head = None
        self.size = 0

    def _len_(self):
        return self.size

    def get_head(self):
        return self.head

    def is_empty(self):
        return self.size == 0

    def display(self):
        if self.size == 0:
            print("No element")
            return
        first = self.head
        print(first.element.element)
        first = first.next
        while first:
            if type(first.element) == type(my_list.head.element):
                print(first.element.element)
                first = first.next
            print(first.element)
            first = first.next

    def reverse_display(self):
        if self.size == 0:
            print("No element")
            return None
        last = my_list.get_tail()
```


PRACTICAL2.py - C:\Users\HP\Documents\DS\PRACTICAL2.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 79 Col: 0

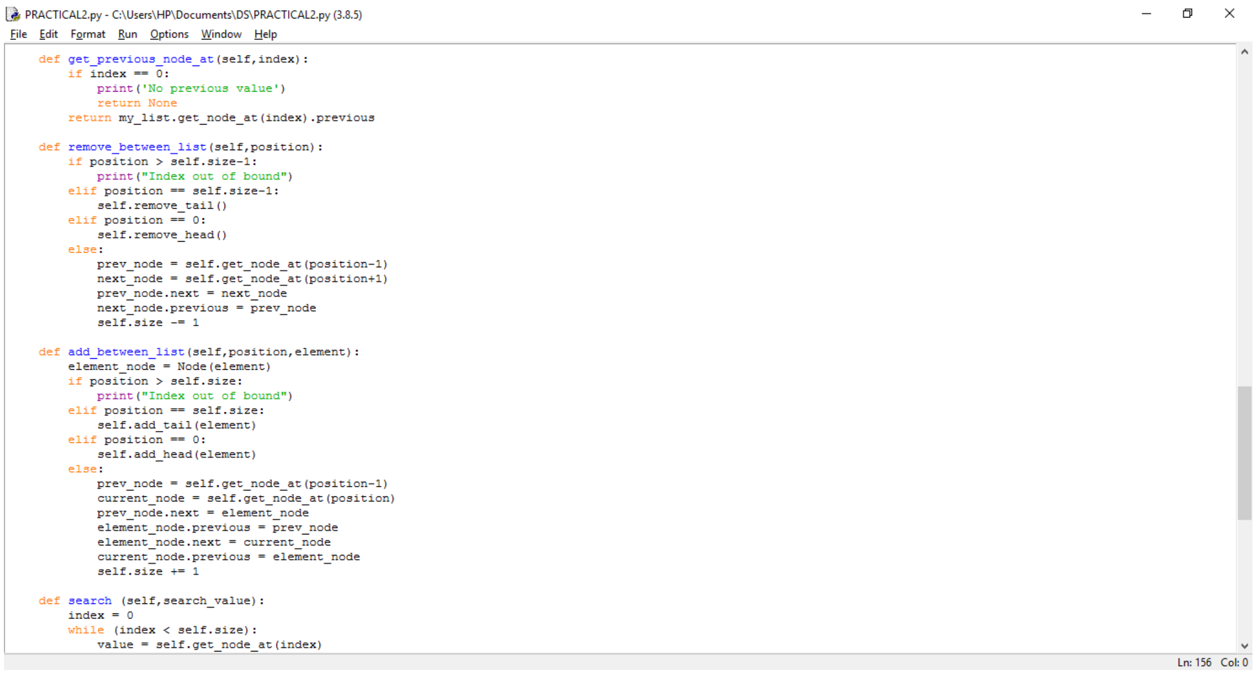
```python
        last = my_list.get_tail()
        print(last.element)
        while last.previous:
            if type(last.previous.element) == type(my_list.head):
                print(last.previous.element.element)
                if last.previous == self.head:
                    return None
                else:
                    last = last.previous
            print(last.previous.element)
            last = last.previous

    def add_head(self,e):
        self.head = Node(e)
        self.size += 1

    def get_tail(self):
        last_object = self.head
        while (last_object.next != None):
            last_object = last_object.next
        return last_object


    def remove_head(self):
        if self.is_empty():
            print("Empty Singly linked list")
        else:
            print("Removing")
            self.head = self.head.next
            self.head.previous = None
            self.size -= 1

    def add_tail(self,e):
        new_value = Node(e)
        new_value.previous = self.get_tail()
        self.get_tail().next = new_value
        self.size += 1

    def find_second_last_element(self):
        if self.size >= 2:
            first = self.head
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**


PRACTICAL2.py - C:\Users\HP\Documents\DS\PRACTICAL2.py (3.8.5)

```python
    def find_second_last_element(self):
        if self.size >= 2:
            first = self.head
            temp_counter = self.size -2
            while temp_counter > 0:
                first = first.next
                temp_counter -= 1
            return first
        else:
            print("Size not sufficient")
        return None

    def remove_tail(self):
        if self.is_empty():
            print("Empty Singly linked list")
        elif self.size == 1:
            self.head == None
            self.size -= 1
        else:
            Node = self.find_second_last_element()
            if Node:
                Node.next = None
                self.size -= 1

    def get_node_at(self,index):
        element_node = self.head
        counter = 0
        if index == 0:
            return element_node.element
        if index > self.size-1:
            print("Index out of bound")
            return None
        while(counter < index):
            element_node = element_node.next
            counter += 1
        return element_node

    def get_previous_node_at(self,index):
        if index == 0:
            print('No previous value')
            return None
```

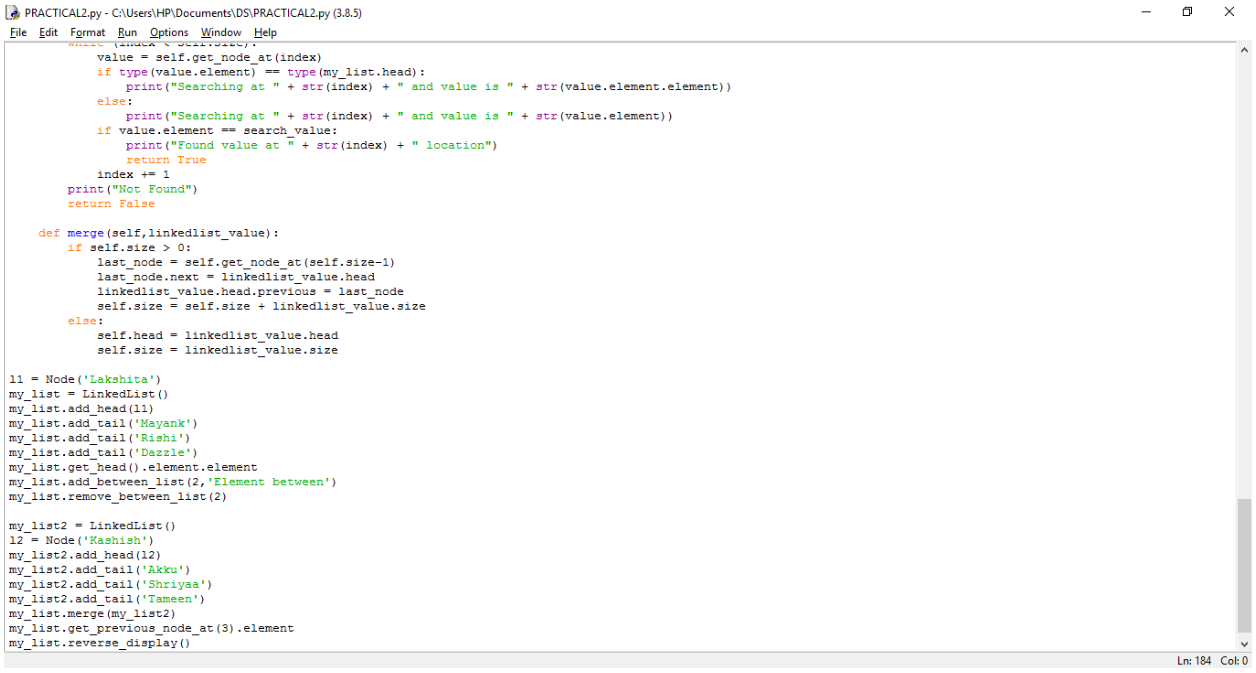
Ln: 119 Col: 0


PRACTICAL2.py - C:\Users\HP\Documents\DS\PRACTICAL2.py (3.8.5)

```python
    def get_previous_node_at(self,index):
        if index == 0:
            print('No previous value')
            return None
        return my_list.get_node_at(index).previous

    def remove_between_list(self,position):
        if position > self.size-1:
            print("Index out of bound")
        elif position == self.size-1:
            self.remove_tail()
        elif position == 0:
            self.remove_head()
        else:
            prev_node = self.get_node_at(position-1)
            next_node = self.get_node_at(position+1)
            prev_node.next = next_node
            next_node.previous = prev_node
            self.size -= 1

    def add_between_list(self,position,element):
        element_node = Node(element)
        if position > self.size:
            print("Index out of bound")
        elif position == self.size:
            self.add_tail(element)
        elif position == 0:
            self.add_head(element)
        else:
            prev_node = self.get_node_at(position-1)
            current_node = self.get_node_at(position)
            prev_node.next = element_node
            element_node.previous = prev_node
            element_node.next = current_node
            current_node.previous = element_node
            self.size += 1

    def search (self,search_value):
        index = 0
        while (index < self.size):
            value = self.get_node_at(index)
```

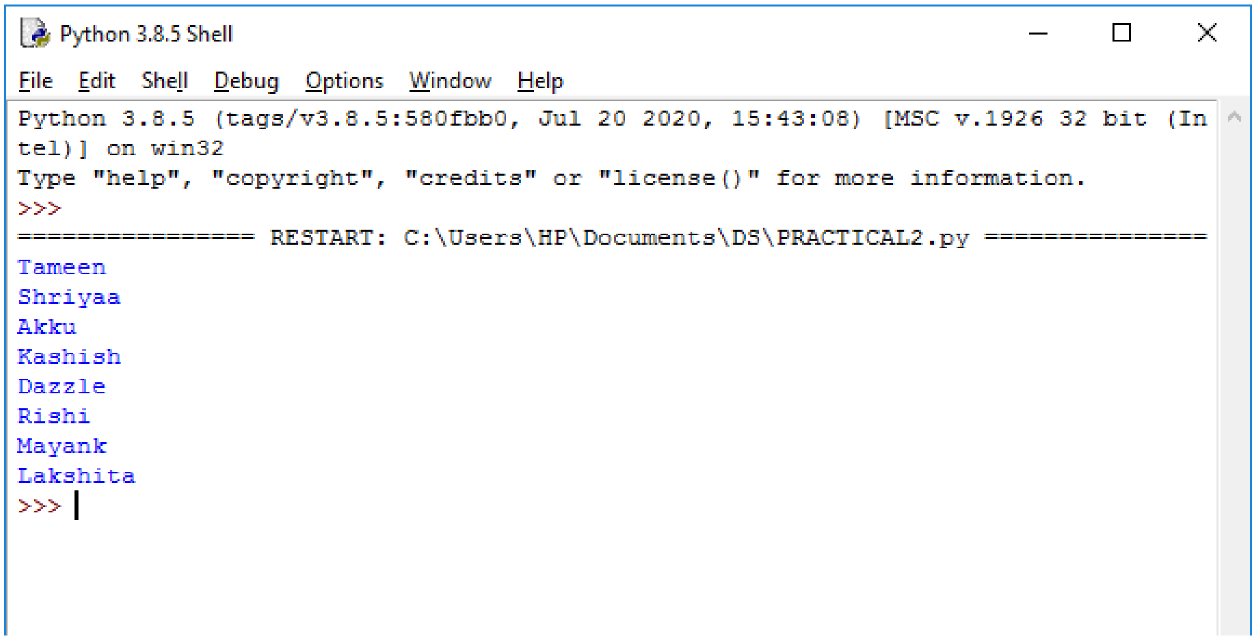
Ln: 156 Col: 0

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**


PRACTICAL2.py - C:\Users\HP\Documents\DS\PRACTICAL2.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 184 Col: 0

```python
        while (index < self.size):
            value = self.get_node_at(index)
            if type(value.element) == type(my_list.head):
                print("Searching at " + str(index) + " and value is " + str(value.element.element))
            else:
                print("Searching at " + str(index) + " and value is " + str(value.element))
            if value.element == search_value:
                print("Found value at " + str(index) + " location")
                return True
            index += 1
        print("Not Found")
        return False

    def merge(self,linkedlist_value):
        if self.size > 0:
            last_node = self.get_node_at(self.size-1)
            last_node.next = linkedlist_value.head
            linkedlist_value.head.previous = last_node
            self.size = self.size + linkedlist_value.size
        else:
            self.head = linkedlist_value.head
            self.size = linkedlist_value.size

l1 = Node('Lakshita')
my_list = LinkedList()
my_list.add_head(l1)
my_list.add_tail('Mayank')
my_list.add_tail('Rishi')
my_list.add_tail('Dazzle')
my_list.get_head().element.element
my_list.add_between_list(2,'Element between')
my_list.remove_between_list(2)

my_list2 = LinkedList()
l2 = Node('Kashish')
my_list2.add_head(l2)
my_list2.add_tail('Akku')
my_list2.add_tail('Shriyaa')
my_list2.add_tail('Tameen')
my_list.merge(my_list2)
my_list.get_previous_node_at(3).element
my_list.reverse_display()
```

OUTPUT -



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\HP\Documents\DS\PRACTICAL2.py ===============
Tameen
Shriyaa
Akku
Kashish
Dazzle
Rishi
Mayank
Lakshita
>>> 
```

Name: LAKSHITA. NARSIAN   Class: SYIT   Rollno: 328

PRACTICAL NO: 3

AIM –

Implement the following for Stack.
3 A : Perform Stack operations using array implementations.

THEORY –

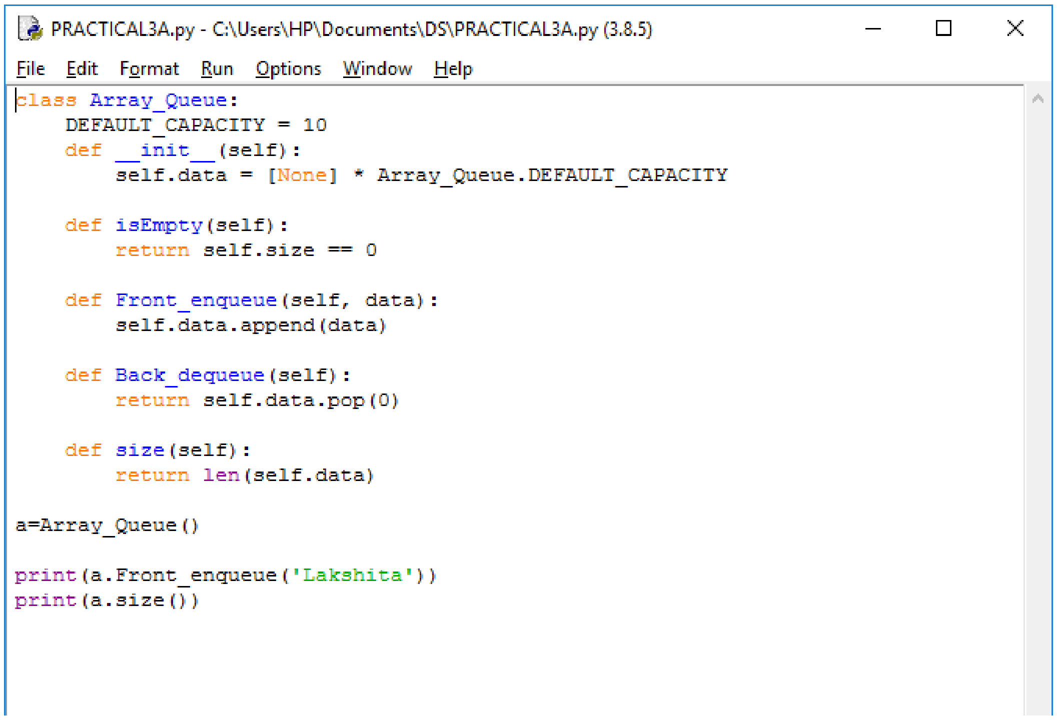
A stack data structure can be implemented using a one-dimensional array. But stack implemented using array stores only a fixed number of data values. This implementation is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable called 'top'. Initially, the top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL3A.py

CODE –

PRACTICAL 3 A:

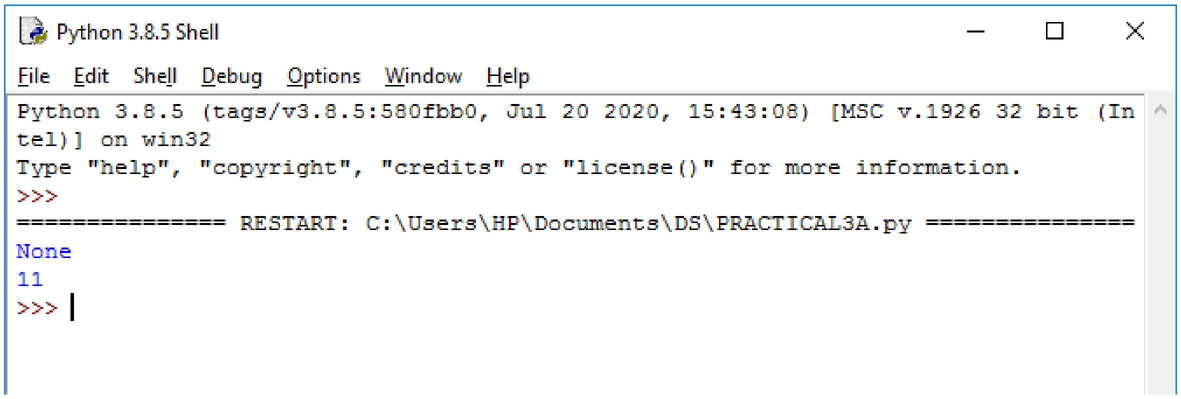

PRACTICAL3A.py - C:\Users\HP\Documents\DS\PRACTICAL3A.py (3.8.5)
File Edit Format Run Options Window Help

```
class Array_Queue:
    DEFAULT_CAPACITY = 10
    def __init__(self):
        self.data = [None] * Array_Queue.DEFAULT_CAPACITY

    def isEmpty(self):
        return self.size == 0

    def Front_enqueue(self, data):
        self.data.append(data)

    def Back_dequeue(self):
        return self.data.pop(0)

    def size(self):
        return len(self.data)

a=Array_Queue()

print(a.Front_enqueue('Lakshita'))
print(a.size())
```

Name: LAKSHITA. NARSIAN   Class: SYIT   Rollno: 328

OUTPUT –

PRACTICAL 3 A:



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL3A.py ===============
None
11
>>> |
```

AIM -

3 B : Implement Tower of Hanoi.

THEORY –

Tower of Hanoi is a mathematical puzzle. It consists of three poles and a number of disks of different sizes which can slide onto any poles. The puzzle starts with the disk in a neat stack in ascending order of size in one pole, the smallest at the top thus making a conical shape. The objective of the puzzle is to move all the disks from one pole (say 'source pole') to another pole (say 'destination pole') with the help of the third pole (say auxiliary pole).

The puzzle has the following two rules:

1. **You can't place a larger disk onto smaller disk**
2. **Only one disk can be moved at a time**

We have also seen that, for n disks, total (2^n)– 1 moves are required.

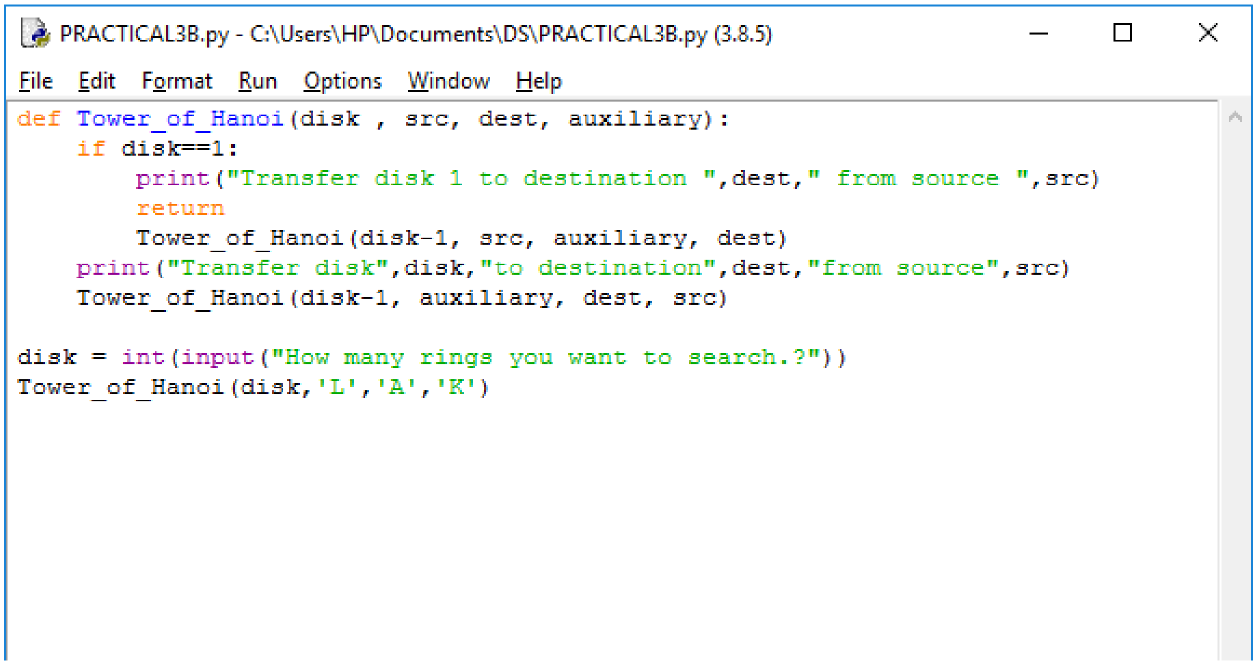
GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL3B.py

CODE –

PRACTICAL 3 B:

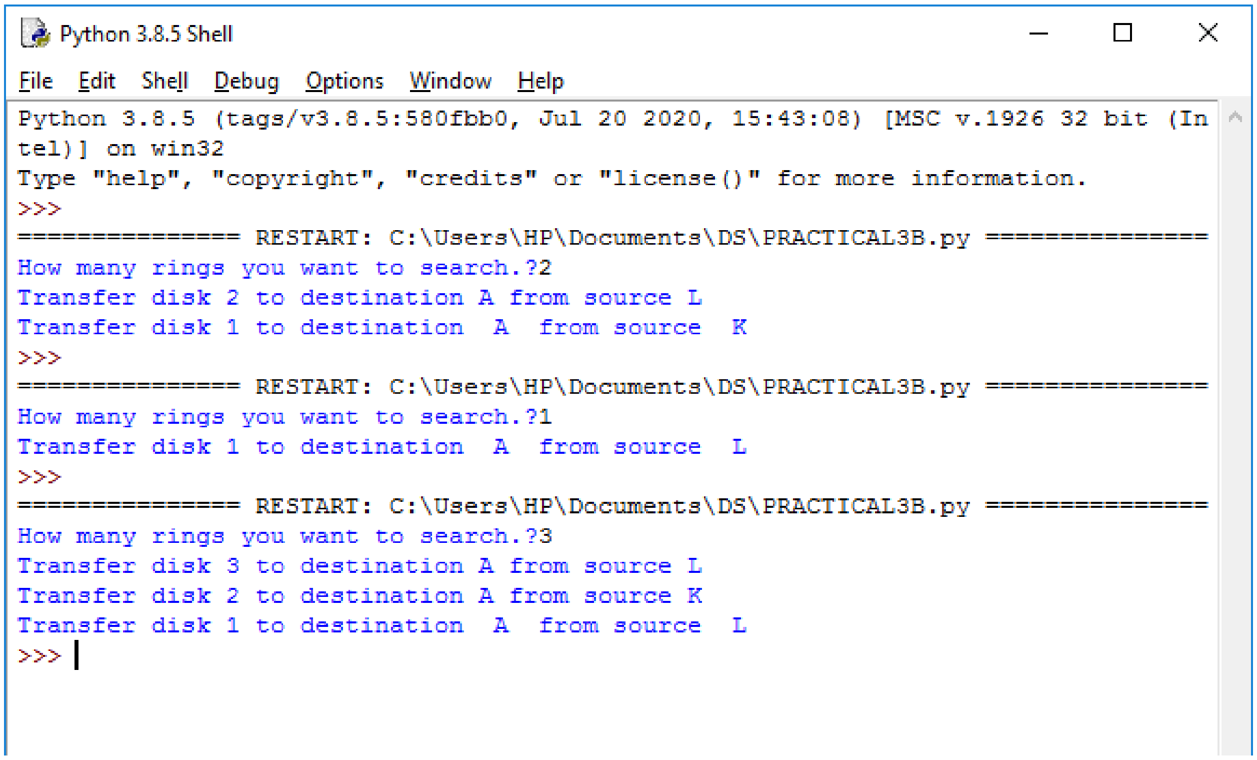
Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**


PRACTICAL3B.py - C:\Users\HP\Documents\DS\PRACTICAL3B.py (3.8.5)
File Edit Format Run Options Window Help

```
def Tower_of_Hanoi(disk , src, dest, auxiliary):
    if disk==1:
        print("Transfer disk 1 to destination ",dest," from source ",src)
        return
        Tower_of_Hanoi(disk-1, src, auxiliary, dest)
    print("Transfer disk",disk,"to destination",dest,"from source",src)
    Tower_of_Hanoi(disk-1, auxiliary, dest, src)

disk = int(input("How many rings you want to search.?"))
Tower_of_Hanoi(disk,'L','A','K')
```

OUTPUT –

PRACTICAL 3 B:



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL3B.py ===============
How many rings you want to search.?2
Transfer disk 2 to destination A from source L
Transfer disk 1 to destination  A  from source  K
>>> 
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL3B.py ===============
How many rings you want to search.?1
Transfer disk 1 to destination  A  from source  L
>>> 
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL3B.py ===============
How many rings you want to search.?3
Transfer disk 3 to destination A from source L
Transfer disk 2 to destination A from source K
Transfer disk 1 to destination  A  from source  L
>>> 
```

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

AIM -

3 C : Write a program to scan a polynomial using linked list and add two polynomials.

THEORY –

A polynomial p(x) is the expression in variable x which is in the form (ax^n + bx^n-1-1-1 + …. + jx + k), where a, b, c …., k fall in the category of real numbers and 'n' is non-negative integer, which is called the degree of polynomial.
An essential characteristic of the polynomial is that each term in the polynomial expression consists of two parts:

- **one is the coefficient**
- **other is the exponent**

Polynomial can be represented in the various ways. These are:

- **By the use of arrays**
- **By the use of Linked List**

In the Polynomial linked list, the coefficients and exponents of the polynomial are defined as the data node of the list.
For adding two polynomials that are stored as a linked list. We need to add the coefficients of variables with the same power. In a linked list node contains 3 members, coefficient value link to the next node.
For adding two polynomials that are represented by a linked list. We check values at the exponent value of the node. For the same values of exponent, we will add the coefficients. For all power, we will check for the coefficients of the exponents that have the same value of exponents and add them. The return the final polynomial.

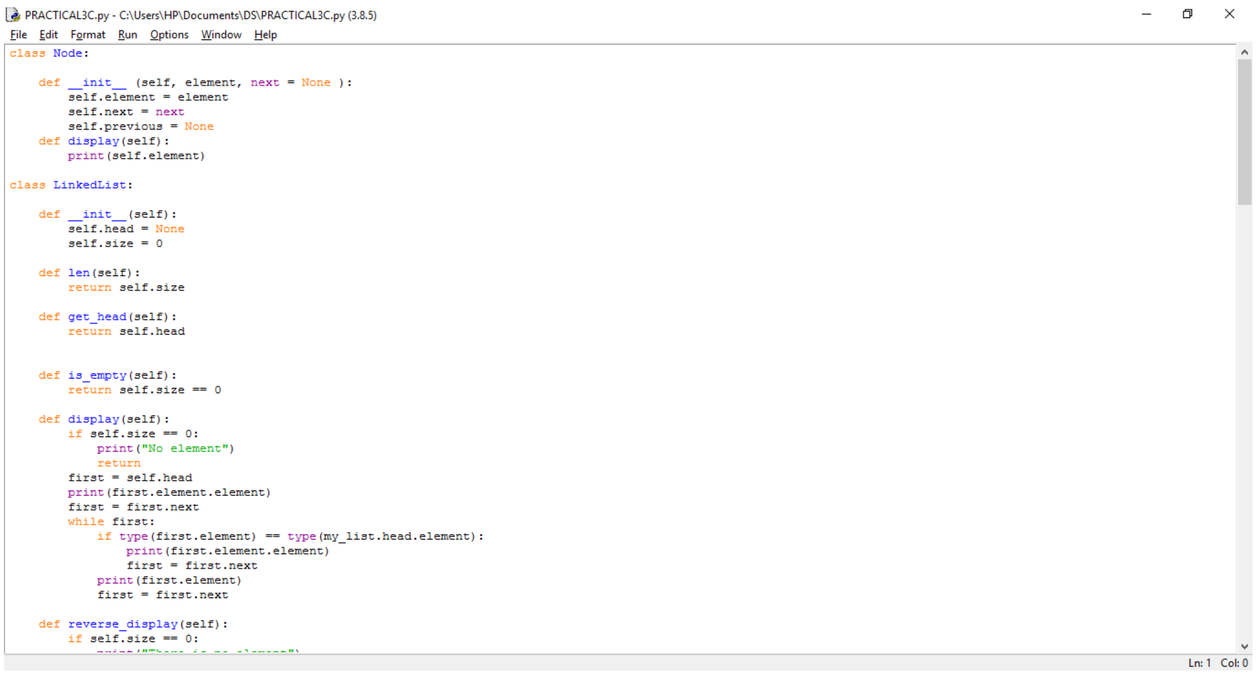
GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL3C.py

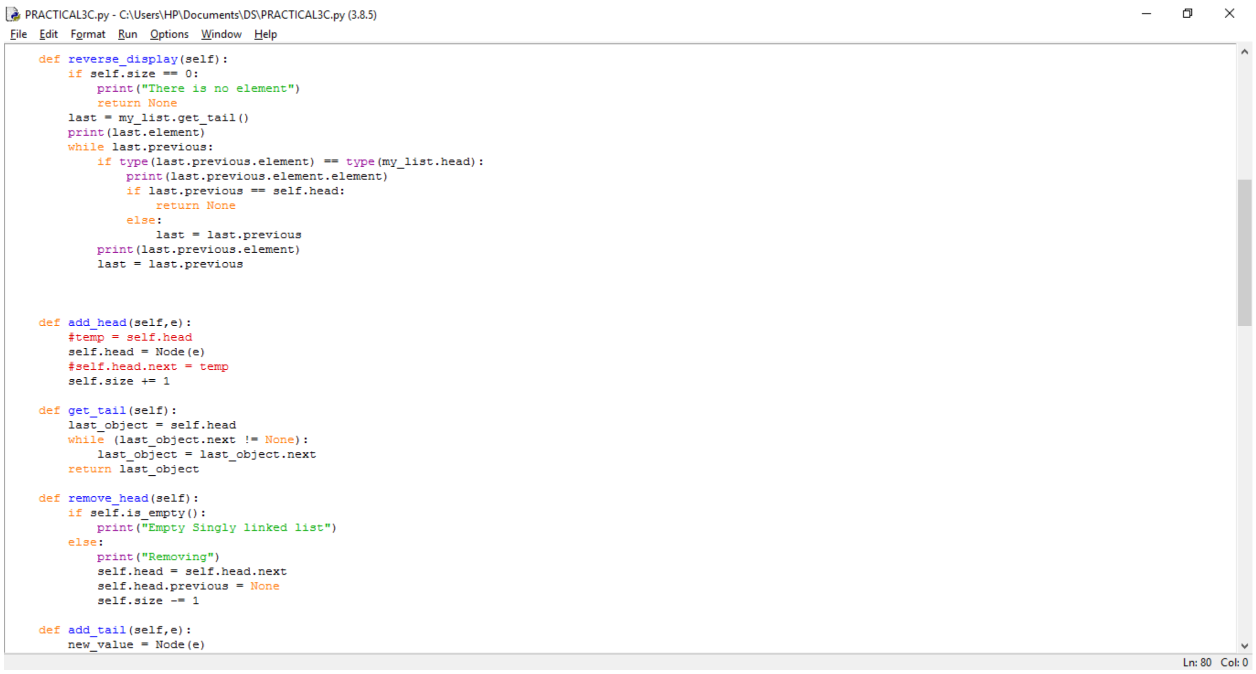
CODE –

PRACTICAL 3 C:

Name: LAKSHITA. NARSIAN　　　Class: SYIT　　　Rollno: 328

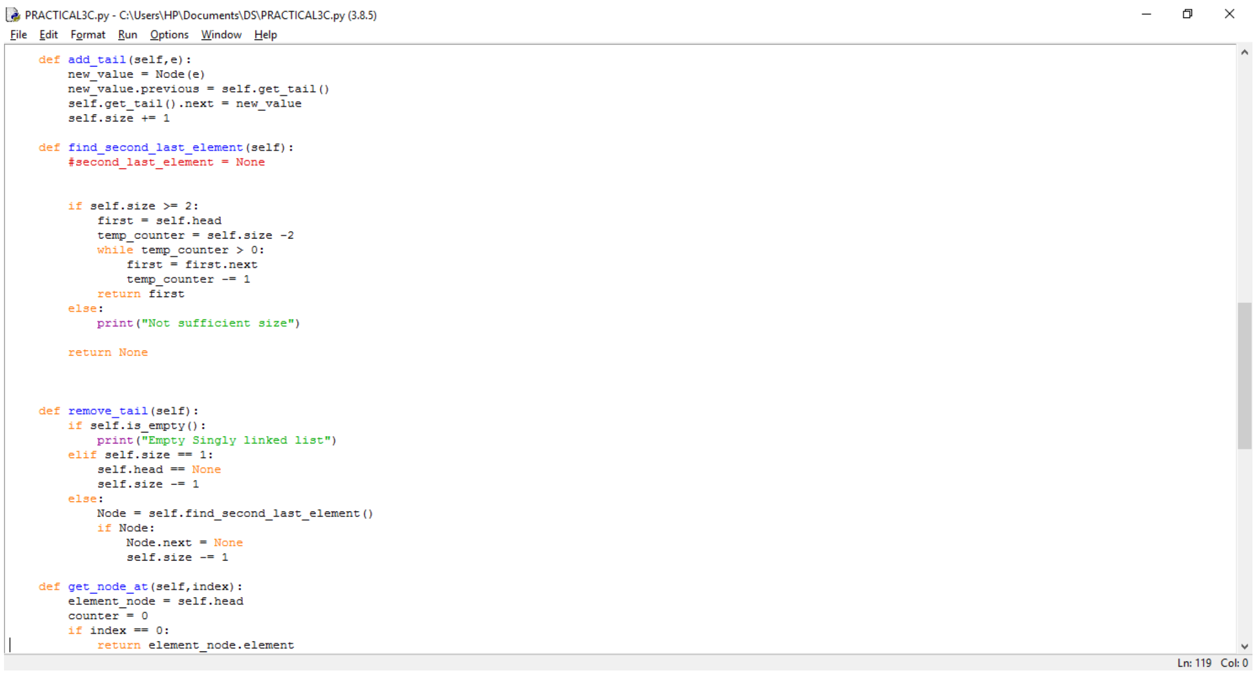

PRACTICAL3C.py - C:\Users\HP\Documents\DS\PRACTICAL3C.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```python
class Node:

    def __init__ (self, element, next = None ):
        self.element = element
        self.next = next
        self.previous = None
    def display(self):
        print(self.element)

class LinkedList:

    def __init__(self):
        self.head = None
        self.size = 0

    def len(self):
        return self.size

    def get_head(self):
        return self.head


    def is_empty(self):
        return self.size == 0

    def display(self):
        if self.size == 0:
            print("No element")
            return
        first = self.head
        print(first.element.element)
        first = first.next
        while first:
            if type(first.element) == type(my_list.head.element):
                print(first.element.element)
                first = first.next
            print(first.element)
            first = first.next

    def reverse_display(self):
        if self.size == 0:
```

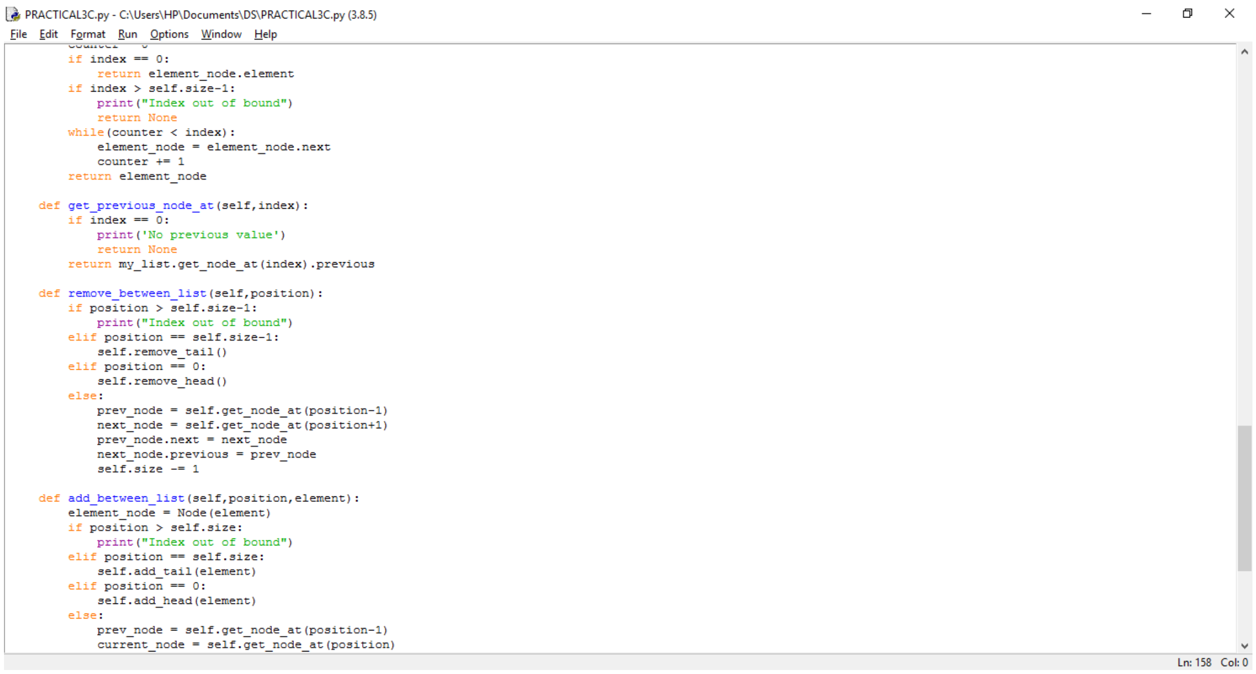

PRACTICAL3C.py - C:\Users\HP\Documents\DS\PRACTICAL3C.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 80 Col: 0

```python
    def reverse_display(self):
        if self.size == 0:
            print("There is no element")
            return None
        last = my_list.get_tail()
        print(last.element)
        while last.previous:
            if type(last.previous.element) == type(my_list.head):
                print(last.previous.element.element)
                if last.previous == self.head:
                    return None
                else:
                    last = last.previous
            print(last.previous.element)
            last = last.previous



    def add_head(self,e):
        #temp = self.head
        self.head = Node(e)
        #self.head.next = temp
        self.size += 1

    def get_tail(self):
        last_object = self.head
        while (last_object.next != None):
            last_object = last_object.next
        return last_object

    def remove_head(self):
        if self.is_empty():
            print("Empty Singly linked list")
        else:
            print("Removing")
            self.head = self.head.next
            self.head.previous = None
            self.size -= 1

    def add_tail(self,e):
        new_value = Node(e)
```

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

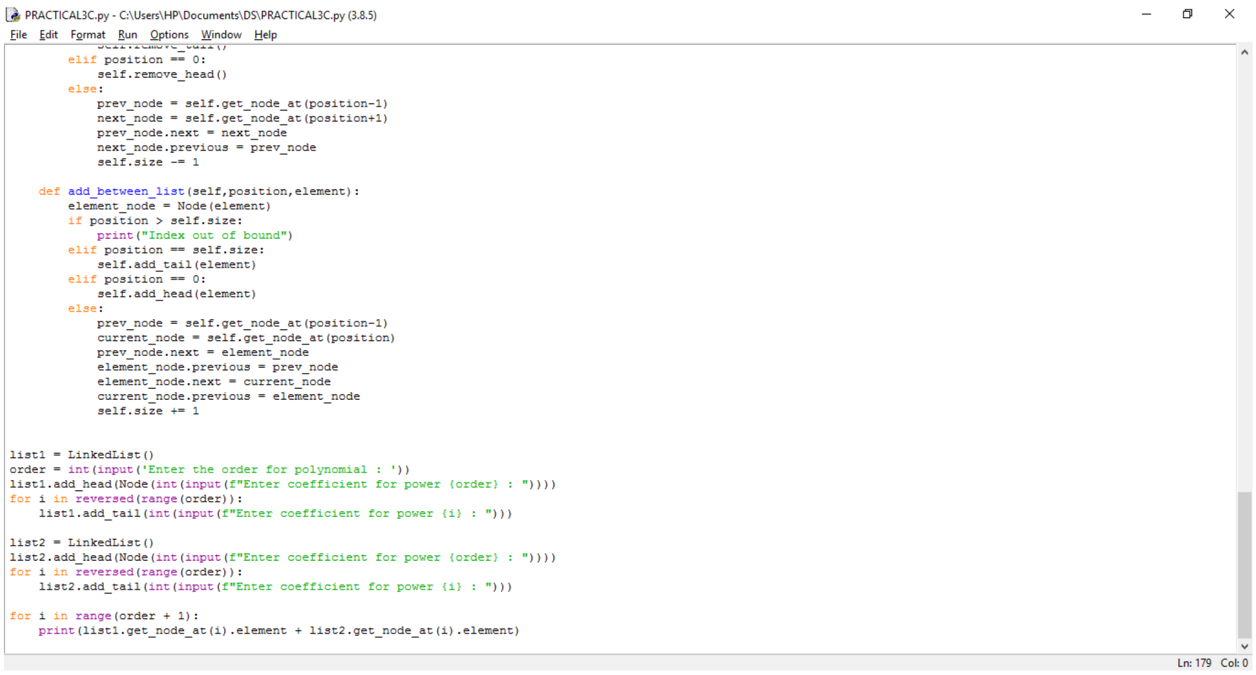

PRACTICAL3C.py - C:\Users\HP\Documents\DS\PRACTICAL3C.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 119 Col: 0

```python
    def add_tail(self,e):
        new_value = Node(e)
        new_value.previous = self.get_tail()
        self.get_tail().next = new_value
        self.size += 1

    def find_second_last_element(self):
        #second_last_element = None


        if self.size >= 2:
            first = self.head
            temp_counter = self.size -2
            while temp_counter > 0:
                first = first.next
                temp_counter -= 1
            return first
        else:
            print("Not sufficient size")

        return None



    def remove_tail(self):
        if self.is_empty():
            print("Empty Singly linked list")
        elif self.size == 1:
            self.head == None
            self.size -= 1
        else:
            Node = self.find_second_last_element()
            if Node:
                Node.next = None
                self.size -= 1

    def get_node_at(self,index):
        element_node = self.head
        counter = 0
        if index == 0:
            return element_node.element
```


PRACTICAL3C.py - C:\Users\HP\Documents\DS\PRACTICAL3C.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 158 Col: 0

```python
        if index == 0:
            return element_node.element
        if index > self.size-1:
            print("Index out of bound")
            return None
        while(counter < index):
            element_node = element_node.next
            counter += 1
        return element_node

    def get_previous_node_at(self,index):
        if index == 0:
            print('No previous value')
            return None
        return my_list.get_node_at(index).previous

    def remove_between_list(self,position):
        if position > self.size-1:
            print("Index out of bound")
        elif position == self.size-1:
            self.remove_tail()
        elif position == 0:
            self.remove_head()
        else:
            prev_node = self.get_node_at(position-1)
            next_node = self.get_node_at(position+1)
            prev_node.next = next_node
            next_node.previous = prev_node
            self.size -= 1

    def add_between_list(self,position,element):
        element_node = Node(element)
        if position > self.size:
            print("Index out of bound")
        elif position == self.size:
            self.add_tail(element)
        elif position == 0:
            self.add_head(element)
        else:
            prev_node = self.get_node_at(position-1)
            current_node = self.get_node_at(position)
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

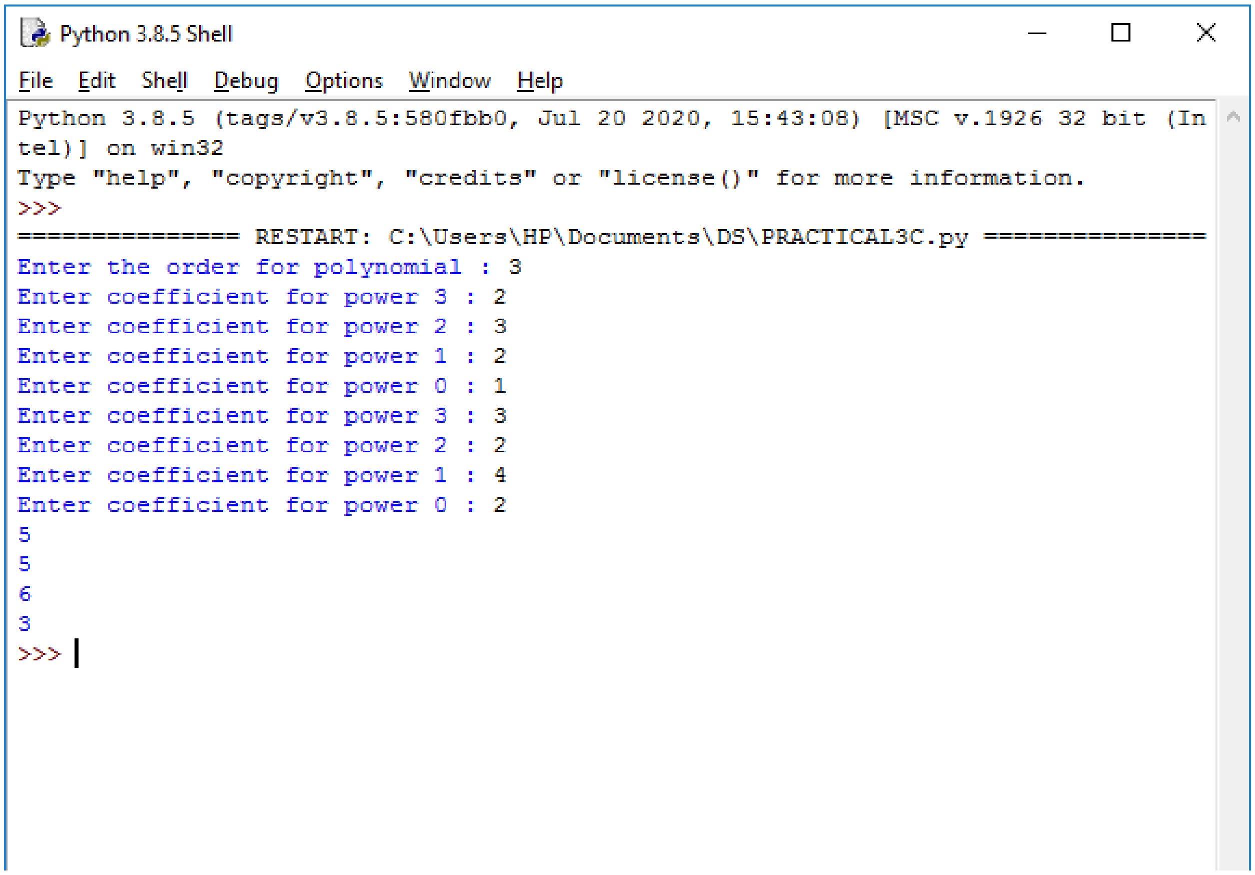

PRACTICAL3C.py - C:\Users\HP\Documents\DS\PRACTICAL3C.py (3.8.5)

```
        elif position == 0:
            self.remove_head()
        else:
            prev_node = self.get_node_at(position-1)
            next_node = self.get_node_at(position+1)
            prev_node.next = next_node
            next_node.previous = prev_node
            self.size -= 1

    def add_between_list(self,position,element):
        element_node = Node(element)
        if position > self.size:
            print("Index out of bound")
        elif position == self.size:
            self.add_tail(element)
        elif position == 0:
            self.add_head(element)
        else:
            prev_node = self.get_node_at(position-1)
            current_node = self.get_node_at(position)
            prev_node.next = element_node
            element_node.previous = prev_node
            element_node.next = current_node
            current_node.previous = element_node
            self.size += 1


list1 = LinkedList()
order = int(input('Enter the order for polynomial : '))
list1.add_head(Node(int(input(f"Enter coefficient for power {order} : "))))
for i in reversed(range(order)):
    list1.add_tail(int(input(f"Enter coefficient for power {i} : ")))

list2 = LinkedList()
list2.add_head(Node(int(input(f"Enter coefficient for power {order} : "))))
for i in reversed(range(order)):
    list2.add_tail(int(input(f"Enter coefficient for power {i} : ")))

for i in range(order + 1):
    print(list1.get_node_at(i).element + list2.get_node_at(i).element)
```

Ln: 179 Col: 0

OUTPUT –

PRACTICAL 3 C:


Python 3.8.5 Shell

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL3C.py ===============
Enter the order for polynomial : 3
Enter coefficient for power 3 : 2
Enter coefficient for power 2 : 3
Enter coefficient for power 1 : 2
Enter coefficient for power 0 : 1
Enter coefficient for power 3 : 3
Enter coefficient for power 2 : 2
Enter coefficient for power 1 : 4
Enter coefficient for power 0 : 2
5
5
6
3
>>> 
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

AIM -

3 D : Write a program to calculate factorial and to compute factors of a given number

I. Using recursion
II. Using iteration.

THEORY –

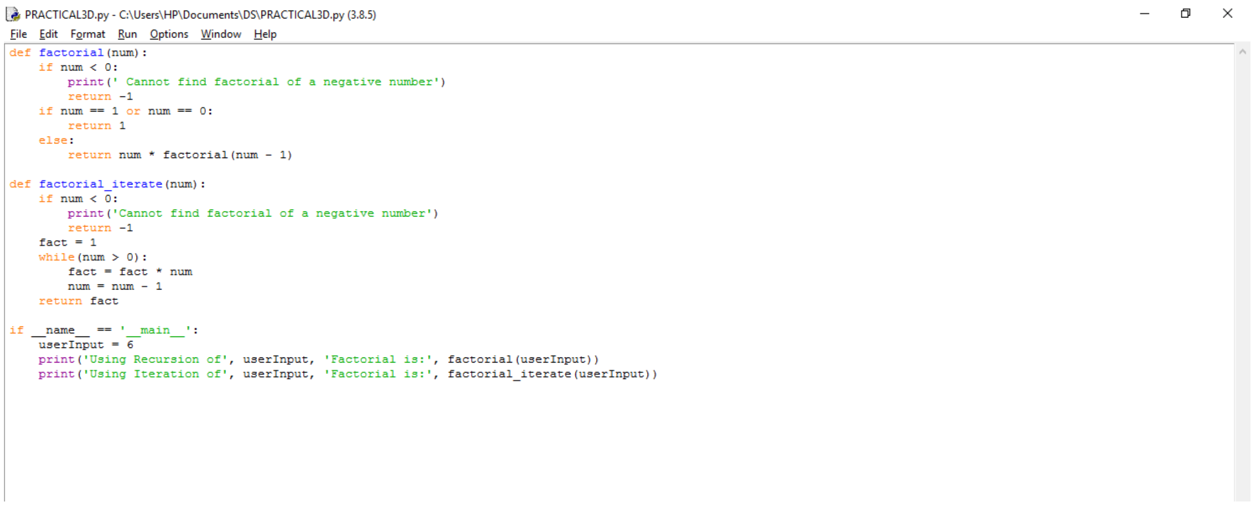
<u>Recursion</u> is the process of defining something in terms of itself.
In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.
When we call this function with a positive integer, it will recursively call itself by decreasing the number.
Each function multiplies the number with the factorial of the number below it until it is equal to one.
<u>Iteration</u>, in the context of computer programming, is a process wherein a set of instructions or structures are repeated in a sequence a specified number of times or until a condition is met. When the first set of instructions is executed again, it is called an iteration.
Factorial of a number is the product of all integers between 1 and itself. To find factorial of a given number, let us form a for loop over a range from 1 to itself. Remember that range() function excludes the stop value. Hence stop value should be one more than the input number. Each number in range is cumulatively multiplied in a variable f which is initialized to 1.

GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL3D.py

CODE –

PRACTICAL 3 D:

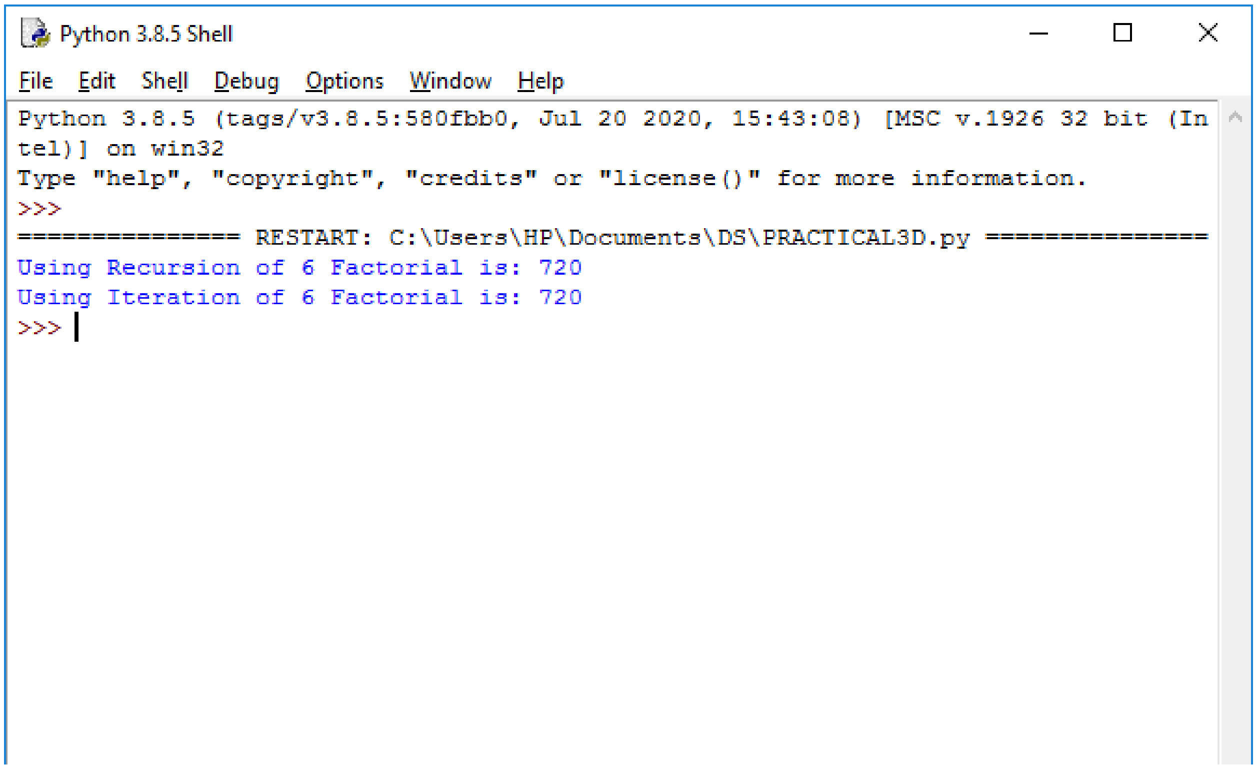

PRACTICAL3D.py - C:\Users\HP\Documents\DS\PRACTICAL3D.py (3.8.5)
File Edit Format Run Options Window Help

```python
def factorial(num):
    if num < 0:
        print(' Cannot find factorial of a negative number')
        return -1
    if num == 1 or num == 0:
        return 1
    else:
        return num * factorial(num - 1)

def factorial_iterate(num):
    if num < 0:
        print('Cannot find factorial of a negative number')
        return -1
    fact = 1
    while(num > 0):
        fact = fact * num
        num = num - 1
    return fact

if __name__ == '__main__':
    userInput = 6
    print('Using Recursion of', userInput, 'Factorial is:', factorial(userInput))
    print('Using Iteration of', userInput, 'Factorial is:', factorial_iterate(userInput))
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

OUTPUT –

PRACTICAL 3 D:



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL3D.py ===============
Using Recursion of 6 Factorial is: 720
Using Iteration of 6 Factorial is: 720
>>> 
```

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

PRACTICAL NO: 4

AIM –

Perform Queues operations using Circular array implementations.

THEORY –

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.
Basic Operations

- **enqueue() - add (store) an item to the queue.**
- **dequeue() - remove (access) an item from the queue**

Few more functions are required to make the above-mentioned queue operation efficient are -

- **peek() - Gets the element at the front of the queue without removing it.**
- **isfull() - Checks if the queue is full.**
- **isempty() - Checks if the queue is empty.**

Operations on a Circular Queue
Enqueue - adding an element in the queue if there is space in the queue.
Dequeue - Removing elements from a queue if there are any elements in the queue
Front- get the first item from the queue.
Rear - get the last item from the queue.
isEmpty / isFull - checks if the queue is empty or full.

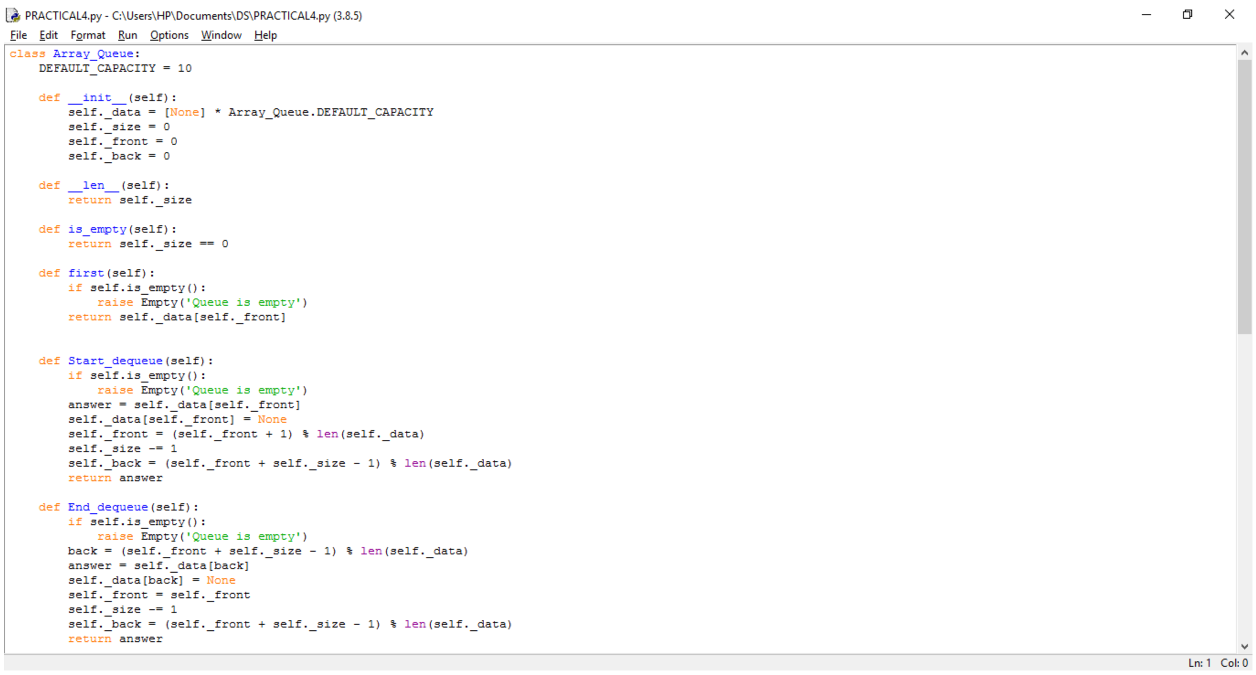
GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL4.py

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

CODE –

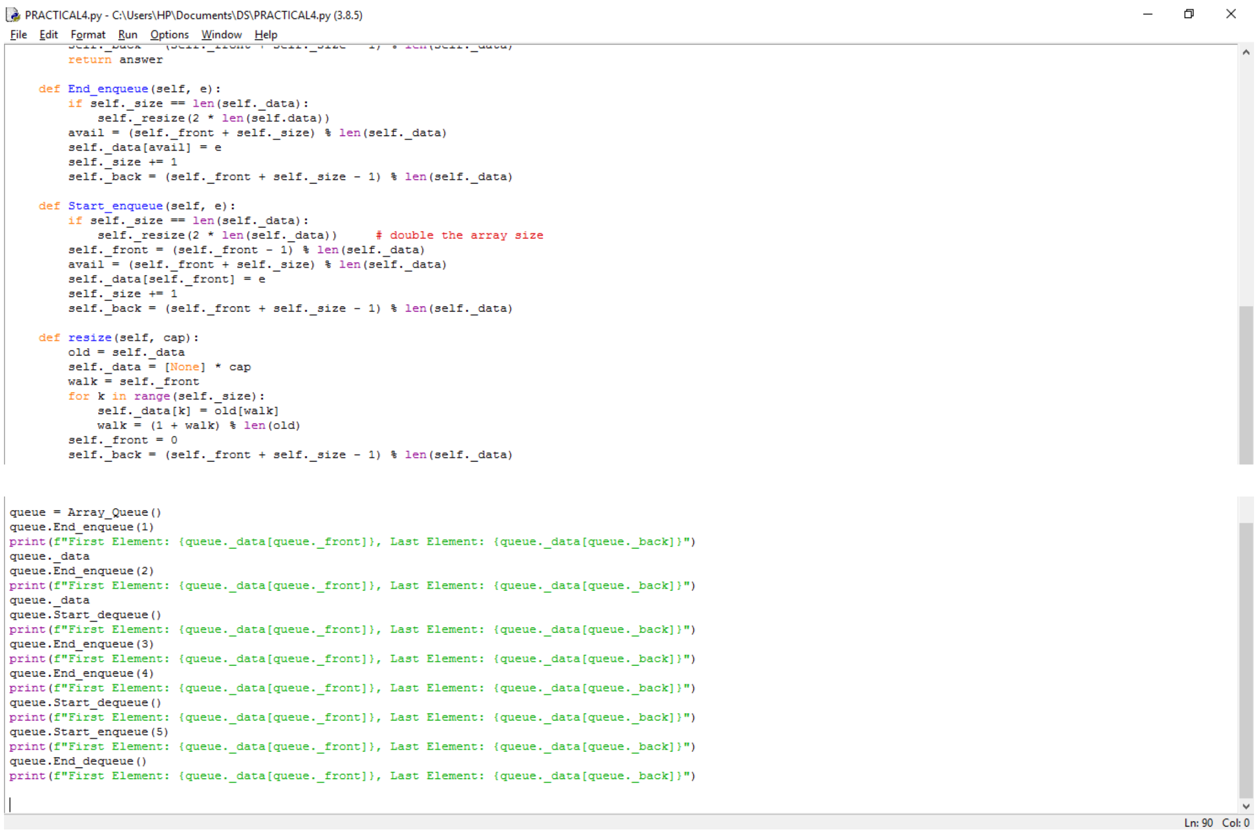
PRACTICAL 4:



PRACTICAL4.py - C:\Users\HP\Documents\DS\PRACTICAL4.py (3.8.5)

```python
class Array_Queue:
    DEFAULT_CAPACITY = 10

    def __init__(self):
        self._data = [None] * Array_Queue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0
        self._back = 0

    def __len__(self):
        return self._size

    def is_empty(self):
        return self._size == 0

    def first(self):
        if self.is_empty():
            raise Empty('Queue is empty')
        return self._data[self._front]


    def Start_dequeue(self):
        if self.is_empty():
            raise Empty('Queue is empty')
        answer = self._data[self._front]
        self._data[self._front] = None
        self._front = (self._front + 1) % len(self._data)
        self._size -= 1
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def End_dequeue(self):
        if self.is_empty():
            raise Empty('Queue is empty')
        back = (self._front + self._size - 1) % len(self._data)
        answer = self._data[back]
        self._data[back] = None
        self._front = self._front
        self._size -= 1
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer
```

Ln: 1 Col: 0



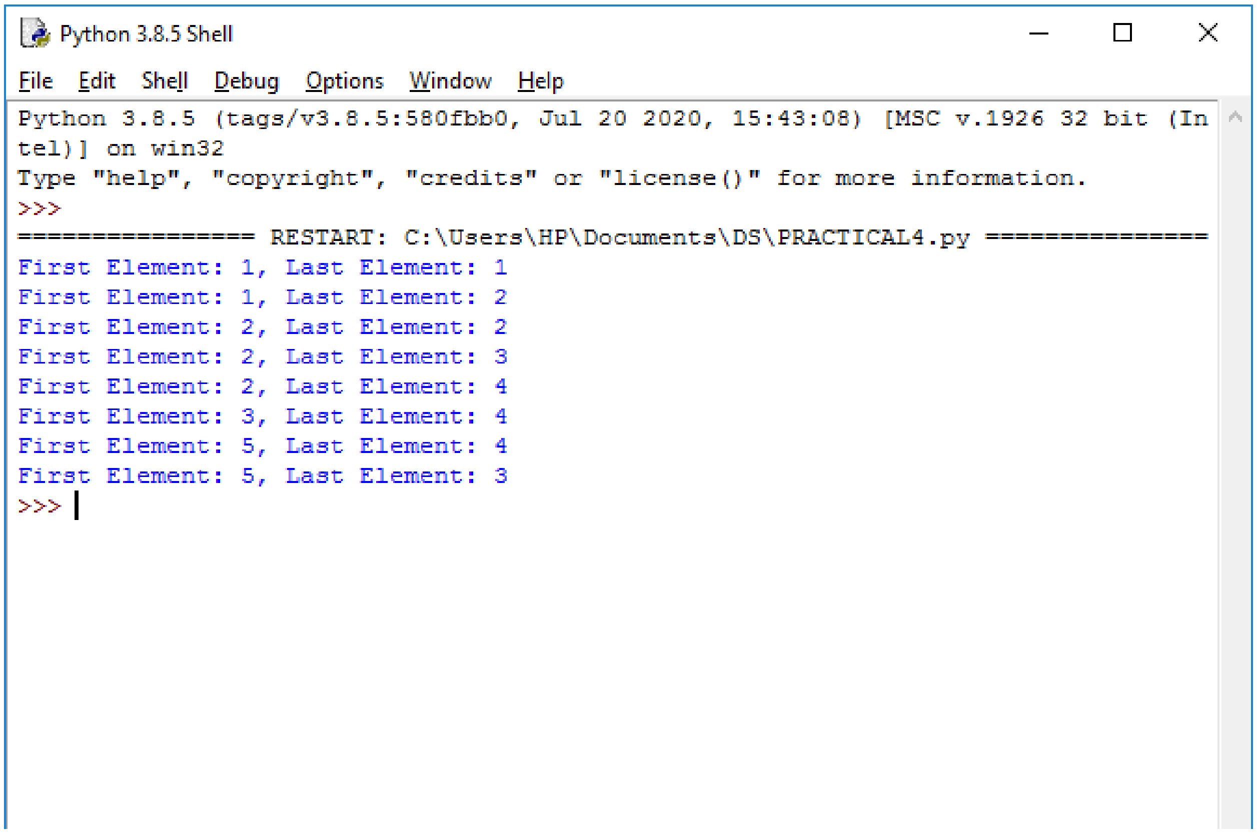
PRACTICAL4.py - C:\Users\HP\Documents\DS\PRACTICAL4.py (3.8.5)

```python
        return answer

    def End_enqueue(self, e):
        if self._size == len(self._data):
            self._resize(2 * len(self.data))
        avail = (self._front + self._size) % len(self._data)
        self._data[avail] = e
        self._size += 1
        self._back = (self._front + self._size - 1) % len(self._data)

    def Start_enqueue(self, e):
        if self._size == len(self._data):
            self._resize(2 * len(self._data))      # double the array size
        self._front = (self._front - 1) % len(self._data)
        avail = (self._front + self._size) % len(self._data)
        self._data[self._front] = e
        self._size += 1
        self._back = (self._front + self._size - 1) % len(self._data)

    def resize(self, cap):
        old = self._data
        self._data = [None] * cap
        walk = self._front
        for k in range(self._size):
            self._data[k] = old[walk]
            walk = (1 + walk) % len(old)
        self._front = 0
        self._back = (self._front + self._size - 1) % len(self._data)


queue = Array_Queue()
queue.End_enqueue(1)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue._data
queue.End_enqueue(2)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue._data
queue.Start_dequeue()
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.End_enqueue(3)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.End_enqueue(4)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.Start_dequeue()
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.Start_enqueue(5)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.End_dequeue()
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
```

Ln: 90 Col: 0

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

OUTPUT –



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
================ RESTART: C:\Users\HP\Documents\DS\PRACTICAL4.py ===============
First Element: 1, Last Element: 1
First Element: 1, Last Element: 2
First Element: 2, Last Element: 2
First Element: 2, Last Element: 3
First Element: 2, Last Element: 4
First Element: 3, Last Element: 4
First Element: 5, Last Element: 4
First Element: 5, Last Element: 3
>>> 
```

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

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 the process of finding a given value position in a list of values. It decides whether a search key is present in the data or not. It is the algorithmic process of finding a particular item in a collection of items.
Searching Techniques :
1. Sequential Search
2. Binary Search

Linear search is a very simple search algorithm. 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 collection

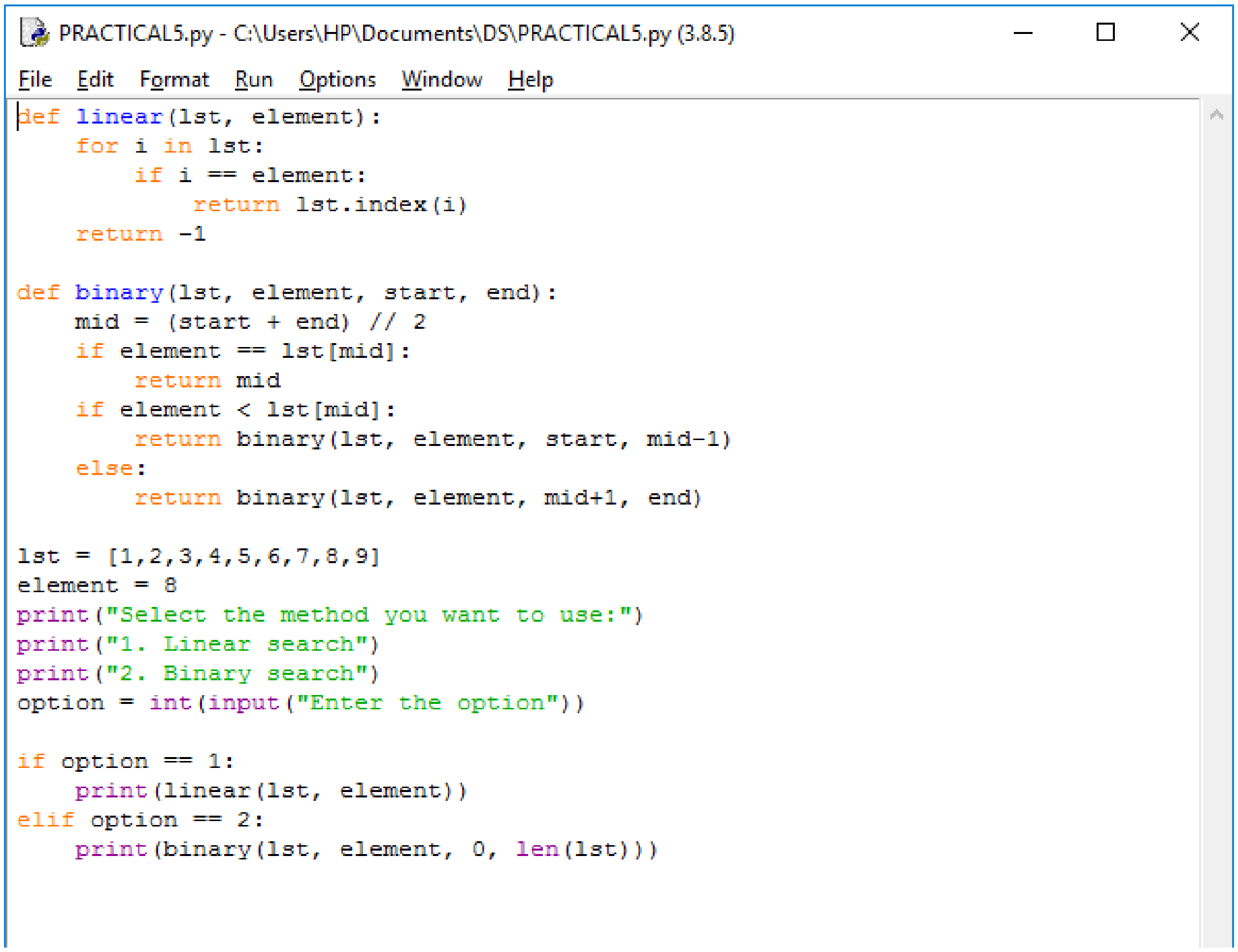
Binary search is a fast search algorithm with run-time complexity of (log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.
Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the sub-array reduces to zero.

GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL5.py

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

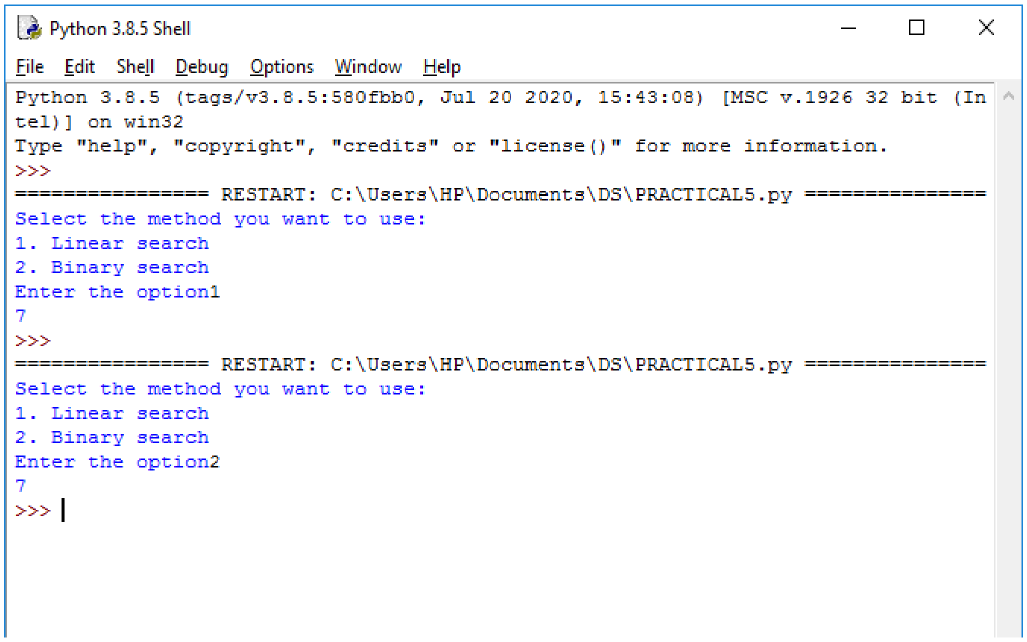
CODE –



PRACTICAL5.py - C:\Users\HP\Documents\DS\PRACTICAL5.py (3.8.5)
File Edit Format Run Options Window Help

```python
def linear(lst, element):
    for i in lst:
        if i == element:
            return lst.index(i)
    return -1

def binary(lst, element, start, end):
    mid = (start + end) // 2
    if element == lst[mid]:
        return mid
    if element < lst[mid]:
        return binary(lst, element, start, mid-1)
    else:
        return binary(lst, element, mid+1, end)

lst = [1,2,3,4,5,6,7,8,9]
element = 8
print("Select the method you want to use:")
print("1. Linear search")
print("2. Binary search")
option = int(input("Enter the option"))

if option == 1:
    print(linear(lst, element))
elif option == 2:
    print(binary(lst, element, 0, len(lst)))
```

OUTPUT –



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\HP\Documents\DS\PRACTICAL5.py ===============
Select the method you want to use:
1. Linear search
2. Binary search
Enter the option1
7
>>>
================ RESTART: C:\Users\HP\Documents\DS\PRACTICAL5.py ===============
Select the method you want to use:
1. Linear search
2. Binary search
Enter the option2
7
>>>
```

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

PRACTICAL NO: 6

AIM –

Write a program to sort 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. 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.

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets.

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort. The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array).

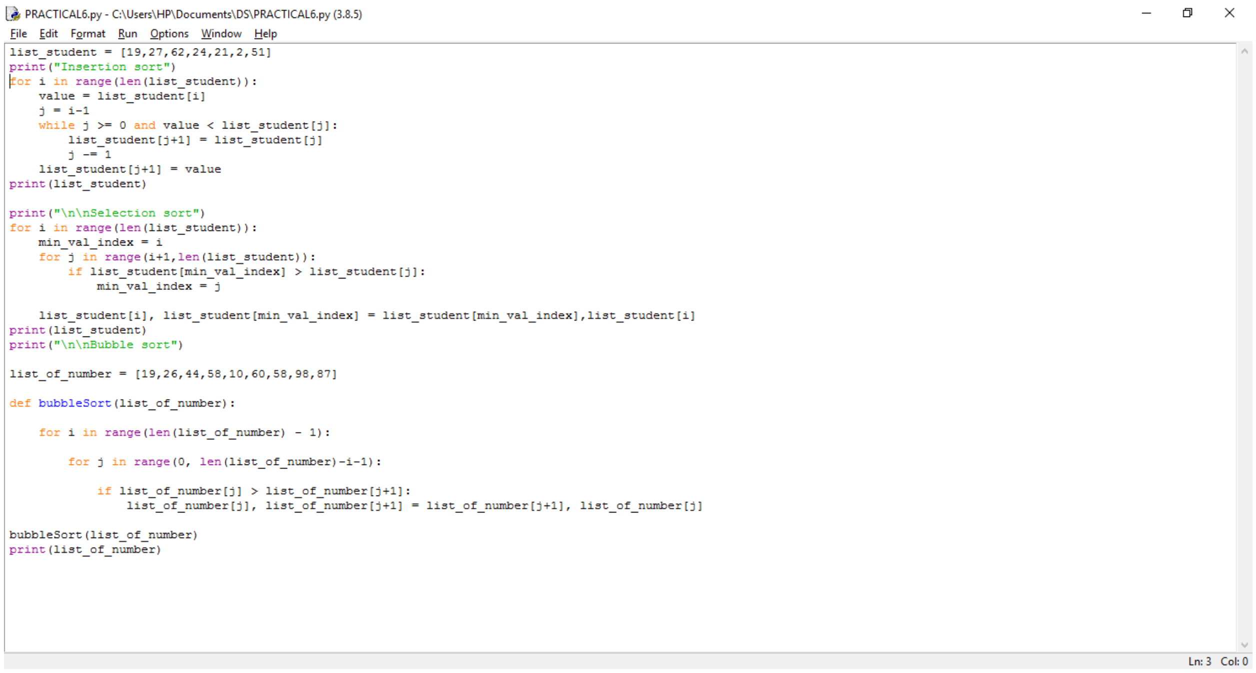
Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list. The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL6.py

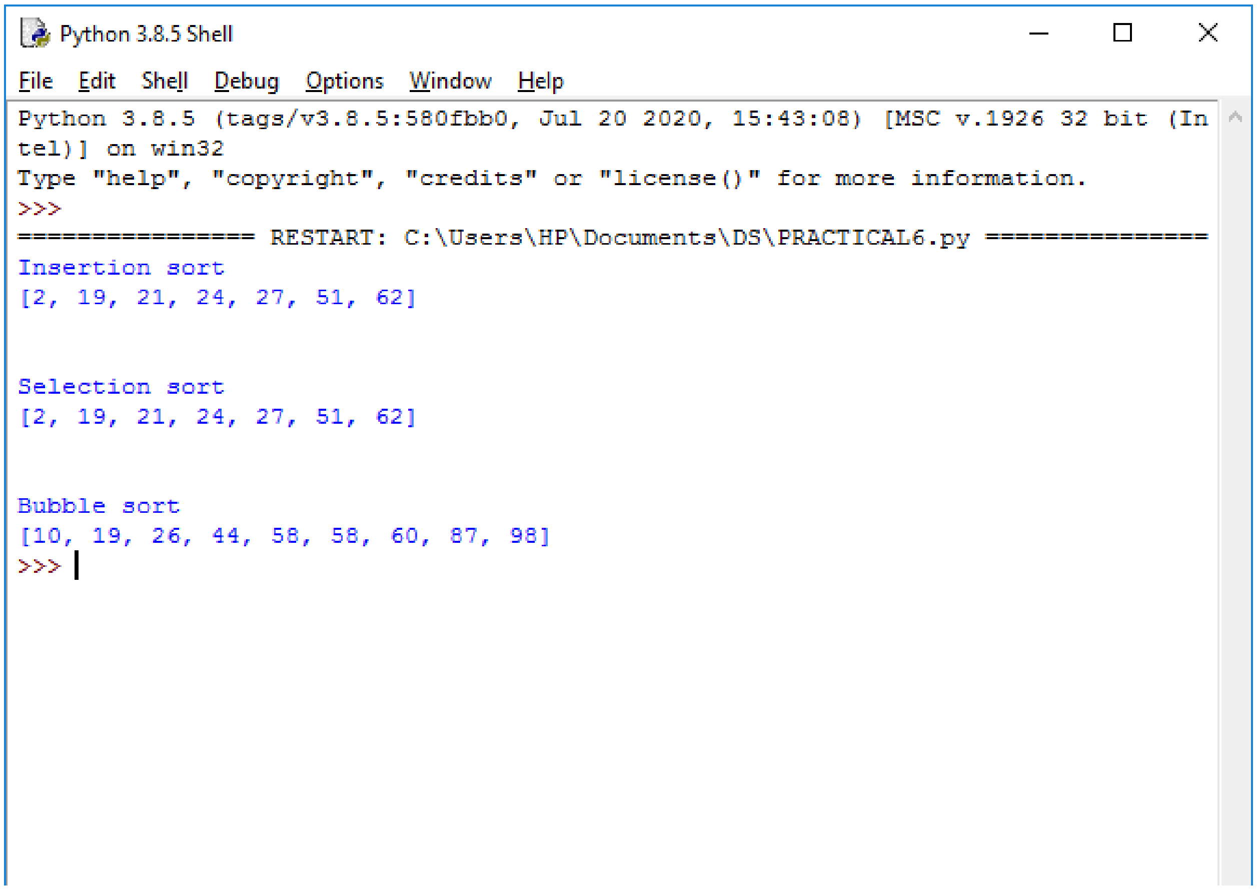
Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

CODE –


PRACTICAL6.py - C:\Users\HP\Documents\DS\PRACTICAL6.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 3 Col: 0

```python
list_student = [19,27,62,24,21,2,51]
print("Insertion sort")
for i in range(len(list_student)):
    value = list_student[i]
    j = i-1
    while j >= 0 and value < list_student[j]:
        list_student[j+1] = list_student[j]
        j -= 1
    list_student[j+1] = value
print(list_student)

print("\n\nSelection sort")
for i in range(len(list_student)):
    min_val_index = i
    for j in range(i+1,len(list_student)):
        if list_student[min_val_index] > list_student[j]:
            min_val_index = j

    list_student[i], list_student[min_val_index] = list_student[min_val_index],list_student[i]
print(list_student)
print("\n\nBubble sort")

list_of_number = [19,26,44,58,10,60,58,98,87]

def bubbleSort(list_of_number):

    for i in range(len(list_of_number) - 1):

        for j in range(0, len(list_of_number)-i-1):

            if list_of_number[j] > list_of_number[j+1]:
                list_of_number[j], list_of_number[j+1] = list_of_number[j+1], list_of_number[j]

bubbleSort(list_of_number)
print(list_of_number)
```

OUTPUT –



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\HP\Documents\DS\PRACTICAL6.py ===============
Insertion sort
[2, 19, 21, 24, 27, 51, 62]


Selection sort
[2, 19, 21, 24, 27, 51, 62]


Bubble sort
[10, 19, 26, 44, 58, 58, 60, 87, 98]
>>> 
```

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

PRACTICAL NO: 7

AIM –

Implement the following for Hashing:
7 A : Write a program to implement the collision technique.

THEORY –

When one or more hash values compete with a single hash table slot, collisions occur. To resolve this, the next available empty slot is assigned to the current hash value. The most common methods are open addressing, chaining, probabilistic hashing, perfect hashing and coalesced hashing technique.
a) Chaining:
This technique implements a linked list and is the most popular collision resolution techniques
b) Open Addressing:
This technique depends on space usage and can be done with linear or quadratic probing techniques. As the name says, this technique tries to find an available slot to store the record. It can be done in one of the 3 ways –

- **Linear probing – Here, the next probe interval is fixed to 1. It supports best caching but miserably fails at clustering.**
- **Quadratic probing – the probe distance is calculated based on the quadratic equation. This is considerably a better option as it balances clustering and caching.**
- **Double hashing – Here, the probing interval is fixed for each record by a second hashing function. This technique has poor cache performance although it does not have any clustering issues.**

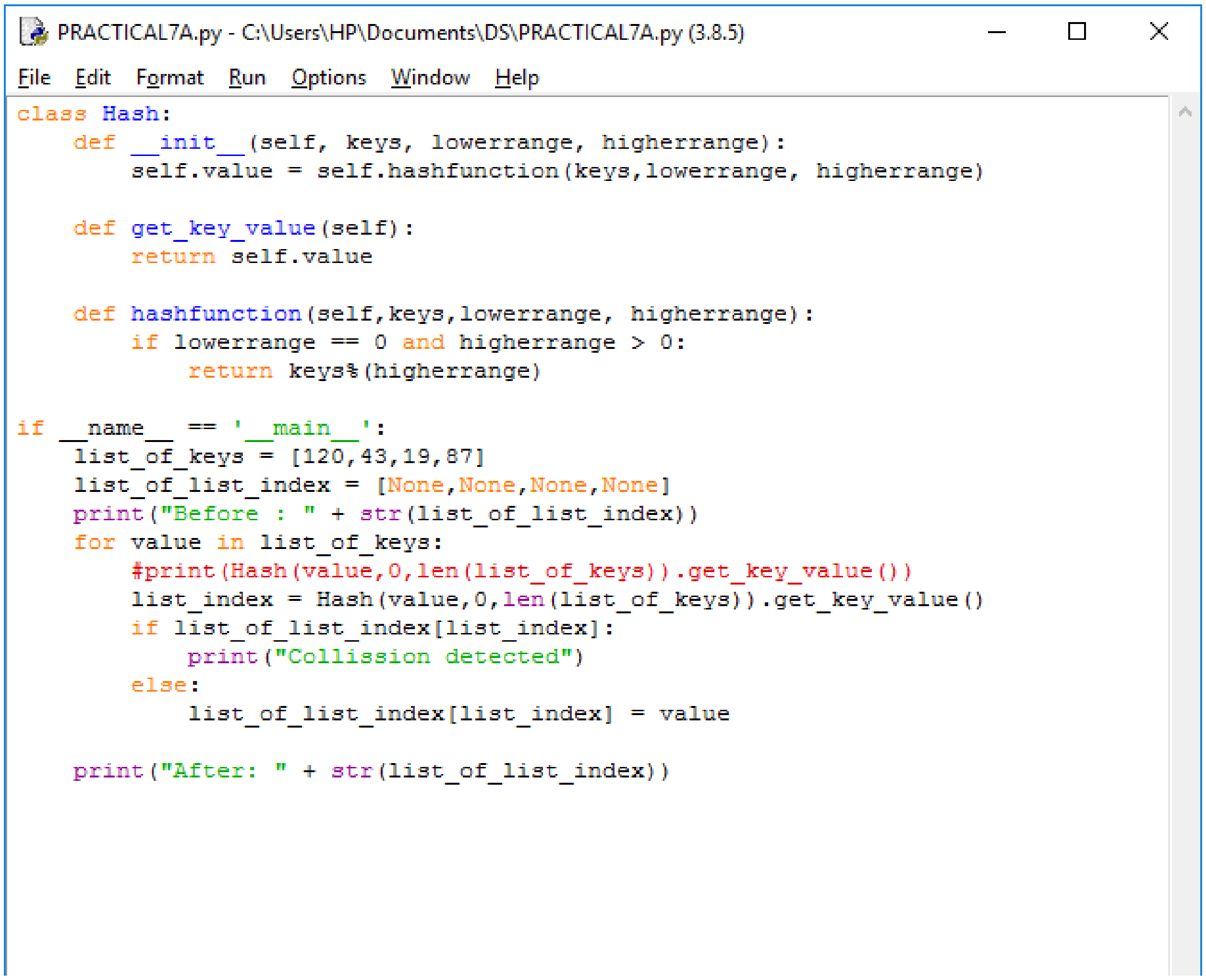
c) Probabilistic hashing:
This is memory based hashing that implements caching. When collision occurs, either the old record is replaced by the new or the new record may be dropped. Although this scenario has a risk of losing data, it is still preferred due to its ease of implementation and high performance.
d) Perfect hashing:
When the slots are uniquely mapped, there is very less chances of collision. However, it can be done where there is a lot of spare memory.
e) Coalesced hashing:
This technique is a combo of open address and chaining methods. A chain of items are stored in the table when there is a collision. The next available table space is used to store the items to prevent collision.

GITHUB URL –

https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL7A.py

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

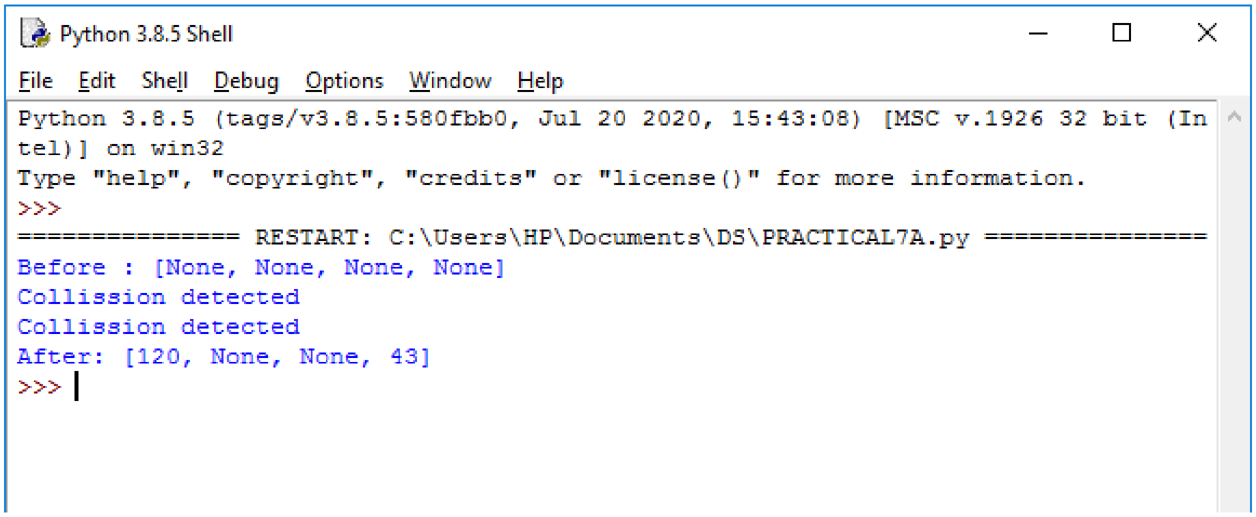
CODE –


PRACTICAL7A.py - C:\Users\HP\Documents\DS\PRACTICAL7A.py (3.8.5)
File Edit Format Run Options Window Help

```
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 = [120,43,19,87]
    list_of_list_index = [None,None,None,None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        #print(Hash(value,0,len(list_of_keys)).get_key_value())
        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))
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

OUTPUT -



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\HP\Documents\DS\PRACTICAL7A.py ===============
Before : [None, None, None, None]
Collission detected
Collission detected
After: [120, None, None, 43]
>>>
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

AIM –

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

THEORY –

Linear probing is a scheme in computer programming for resolving collisions in hash tables, data structures for maintaining a collection of key–value pairs and looking up the value associated with a given key.
Challenges in Linear Probing:
Primary Clustering: One of the problems with linear probing is Primary clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search an element.

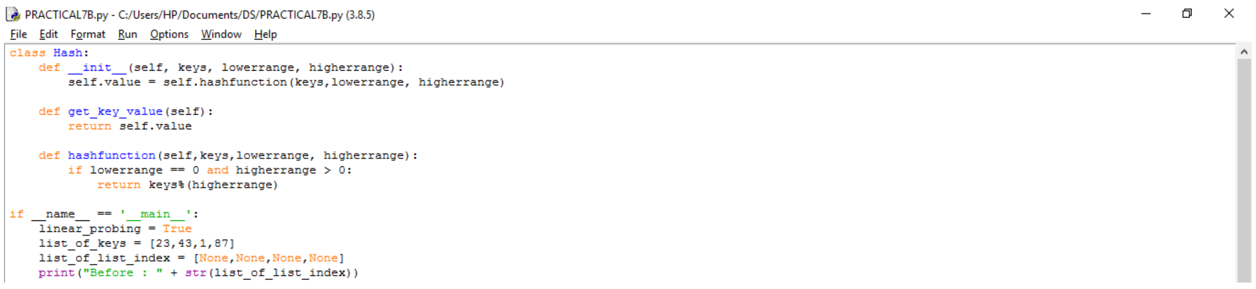
Secondary Clustering: Secondary clustering is less severe, two records do only have the same collision chain (Probe Sequence) if their initial position is the same.
Quadratic Probing: We look for i2‘th slot in i’th iteration.
Double Hashing: We use another hash function hash2(x) and look for i*hash2(x) slot in i’th rotation.

Comparison of above three:
Linear probing has the best cache performance but suffers from clustering. One more advantage of Linear probing is easy to compute. Quadratic probing lies between the two in terms of cache performance and clustering. Double hashing has poor cache performance but no clustering. Double hashing requires more computation time as two hash functions need to be computed.

GITHUB URL –

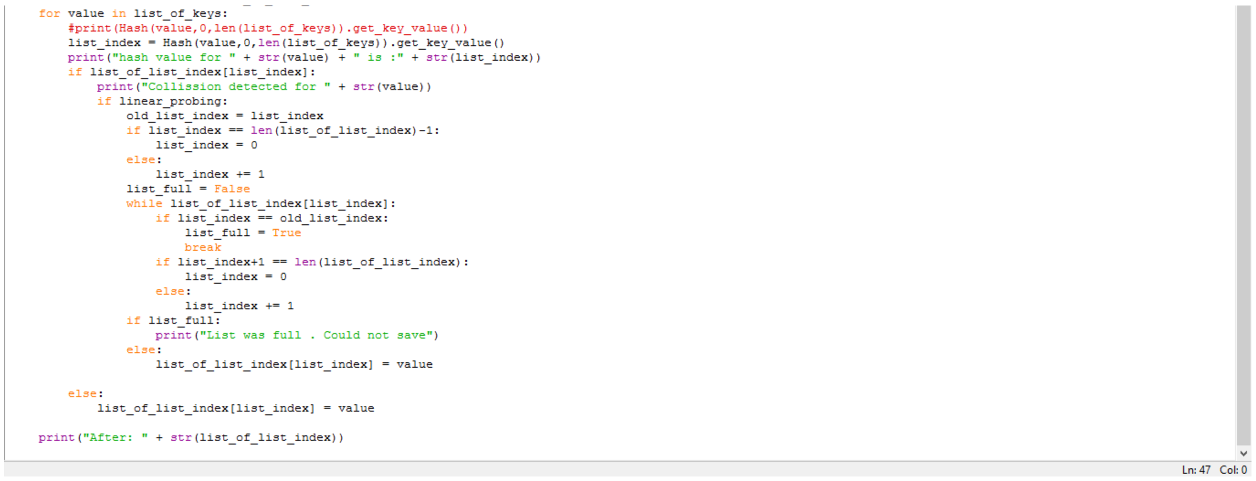
https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL7B.py

CODE –



PRACTICAL7B.py - C:/Users/HP/Documents/DS/PRACTICAL7B.py (3.8.5)

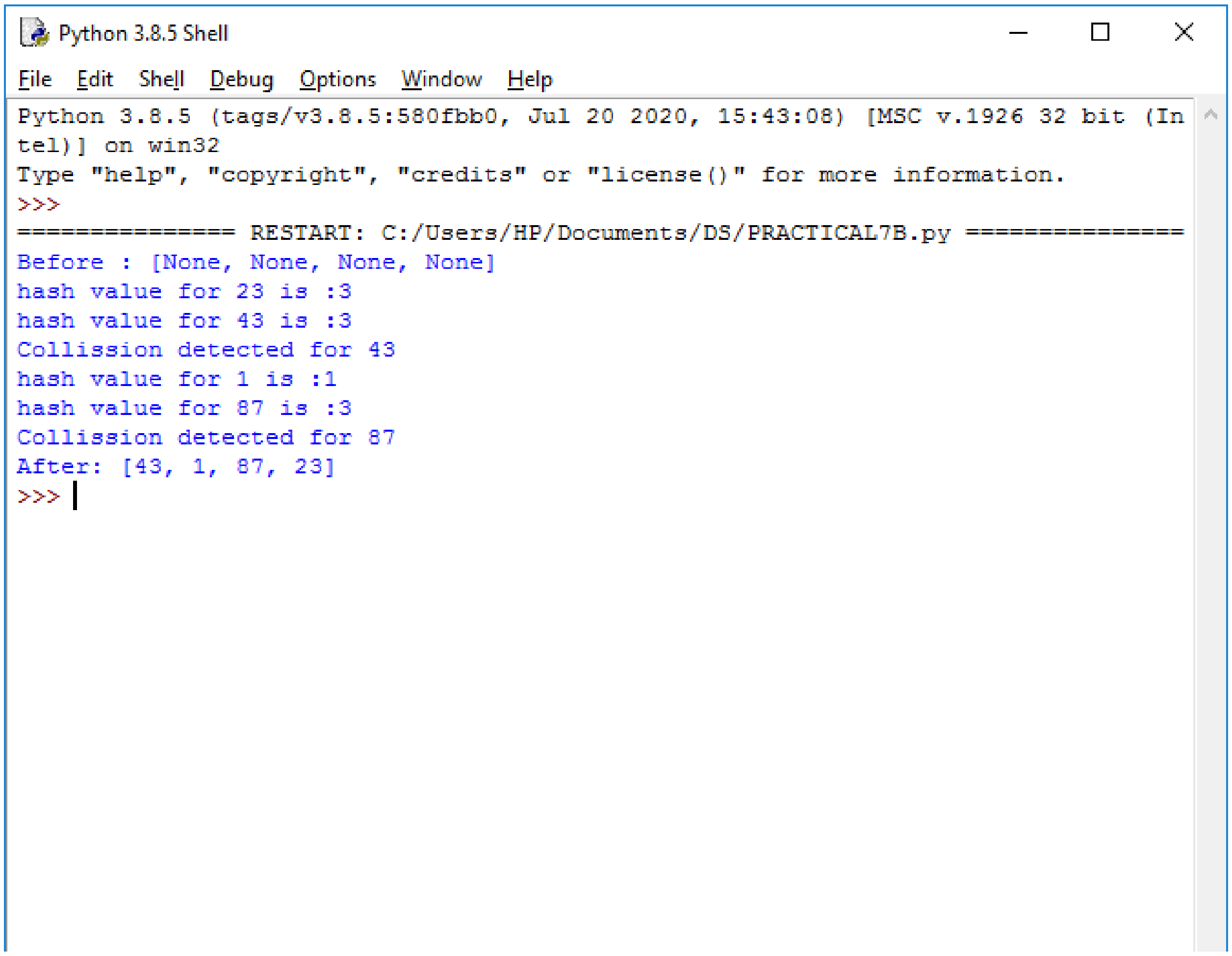
```
class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys,lowerrange, higherrange)

    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__':
    linear_probing = True
    list_of_keys = [23,43,1,87]
    list_of_list_index = [None,None,None,None]
    print("Before : " + str(list_of_list_index))
```



```python
    for value in list_of_keys:
        #print(Hash(value,0,len(list_of_keys)).get_key_value())
        list_index = Hash(value,0,len(list_of_keys)).get_key_value()
        print("hash value for " + str(value) + " is :" + str(list_index))
        if list_of_list_index[list_index]:
            print("Collission detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index)-1:
                    list_index = 0
                else:
                    list_index += 1
                list_full = False
                while list_of_list_index[list_index]:
                    if list_index == old_list_index:
                        list_full = True
                        break
                    if list_index+1 == len(list_of_list_index):
                        list_index = 0
                    else:
                        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: 47 Col: 0

OUTPUT –



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
=============== RESTART: C:/Users/HP/Documents/DS/PRACTICAL7B.py ===============
Before : [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 :3
Collission detected for 87
After: [43, 1, 87, 23]
>>> 
```

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

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:**

1. **Inorder (Left, Root, Right)**
2. **Preorder (Root, Left, Right)**
3. **Postorder (Left, Right, Root)**

- **Breadth First or Level Order Traversal**

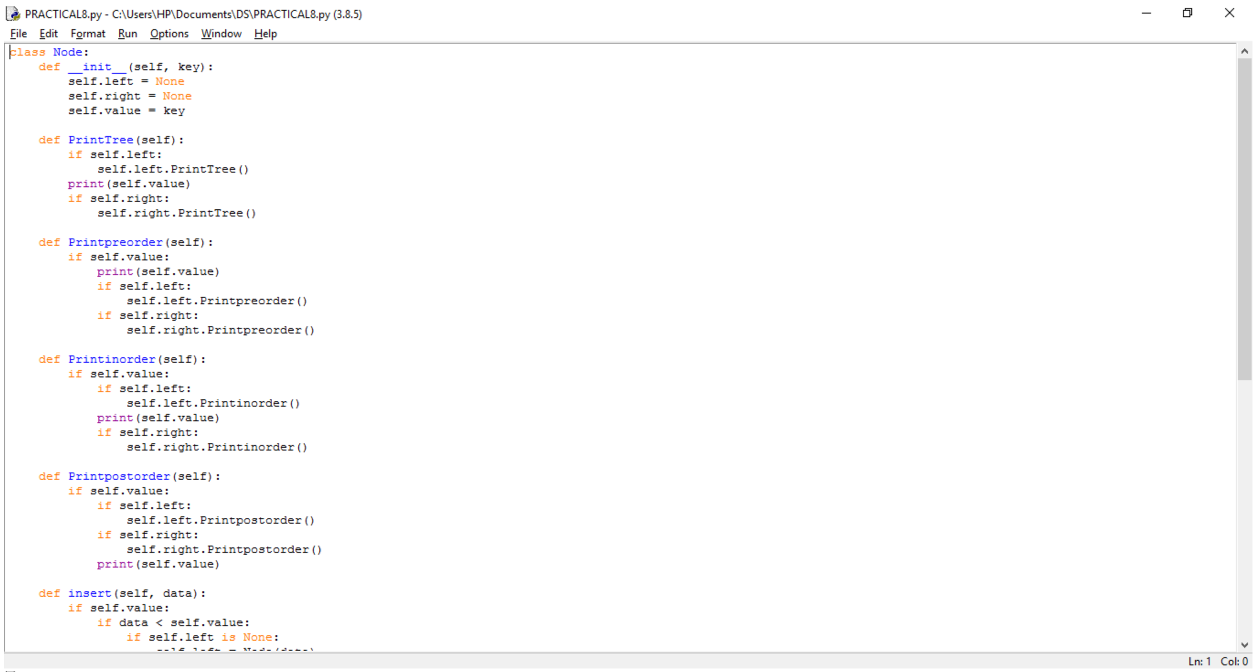
Uses of Inorder:
In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal s reversed can be used.
Uses of Preorder:
Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.
Uses of Postorder:
Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree.

GITHUB URL –

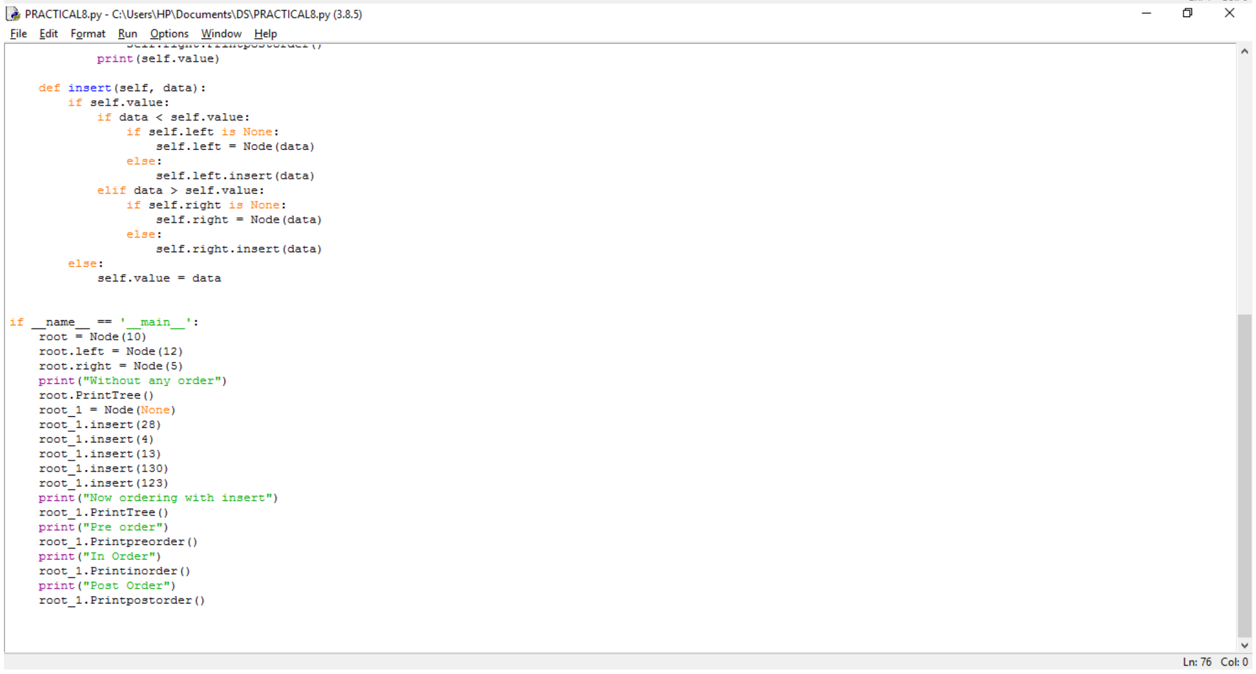
https://github.com/LakshitaNarsian/DS/blob/master/PRACTICAL8.py

Name: LAKSHITA. NARSIAN Class: SYIT Rollno: 328

CODE –


PRACTICAL8.py - C:\Users\HP\Documents\DS\PRACTICAL8.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

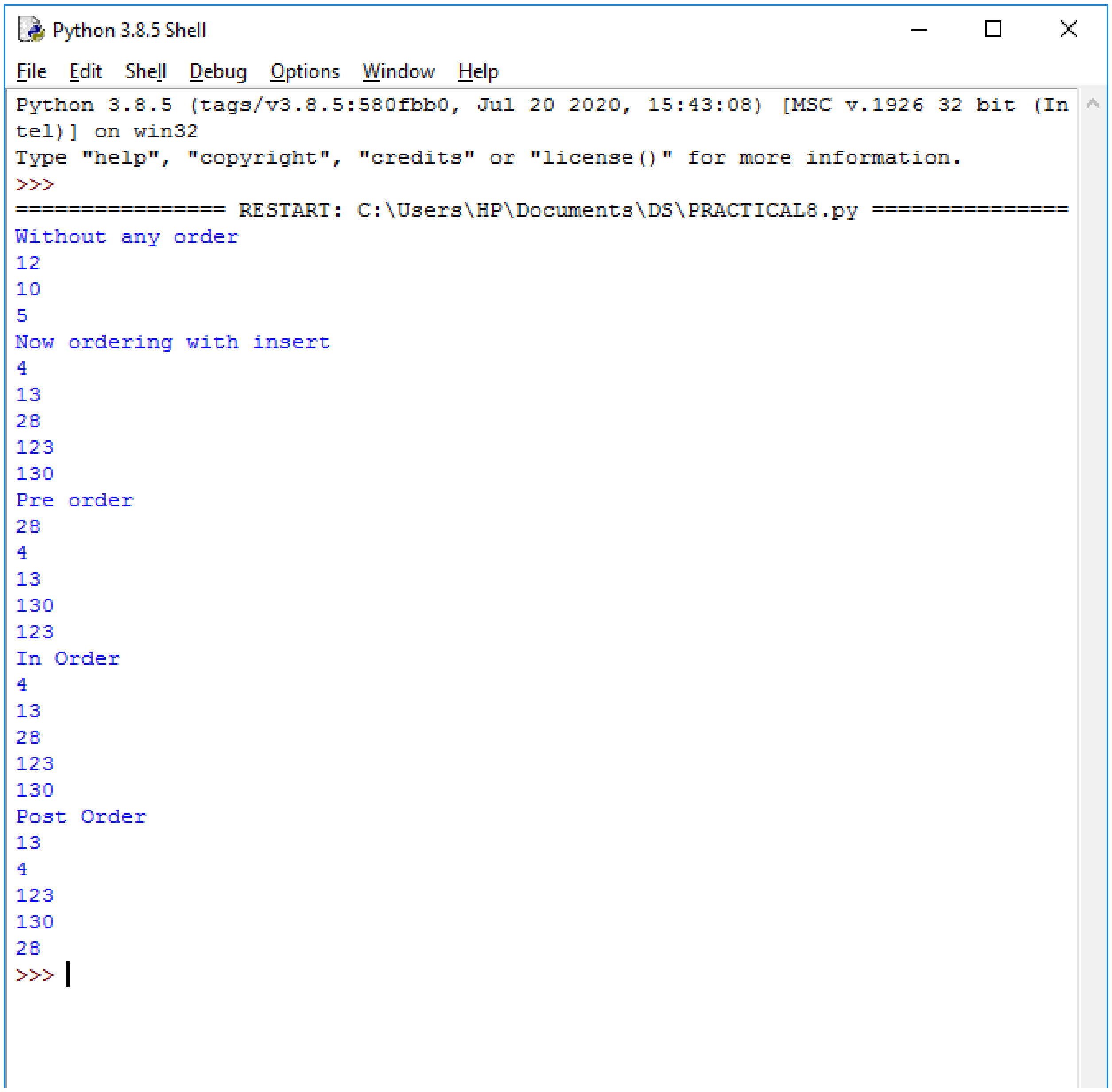
```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.value = key

    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.value)
        if self.right:
            self.right.PrintTree()

    def Printpreorder(self):
        if self.value:
            print(self.value)
            if self.left:
                self.left.Printpreorder()
            if self.right:
                self.right.Printpreorder()

    def Printinorder(self):
        if self.value:
            if self.left:
                self.left.Printinorder()
            print(self.value)
            if self.right:
                self.right.Printinorder()

    def Printpostorder(self):
        if self.value:
            if self.left:
                self.left.Printpostorder()
            if self.right:
                self.right.Printpostorder()
            print(self.value)

    def insert(self, data):
        if self.value:
            if data < self.value:
                if self.left is None:
```


PRACTICAL8.py - C:\Users\HP\Documents\DS\PRACTICAL8.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 76 Col: 0

```
            print(self.value)

    def insert(self, data):
        if self.value:
            if data < self.value:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.value:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.value = data


if __name__ == '__main__':
    root = Node(10)
    root.left = Node(12)
    root.right = Node(5)
    print("Without any order")
    root.PrintTree()
    root_1 = Node(None)
    root_1.insert(28)
    root_1.insert(4)
    root_1.insert(13)
    root_1.insert(130)
    root_1.insert(123)
    print("Now ordering with insert")
    root_1.PrintTree()
    print("Pre order")
    root_1.Printpreorder()
    print("In Order")
    root_1.Printinorder()
    print("Post Order")
    root_1.Printpostorder()
```

Name: LAKSHITA. NARSIAN **Class: SYIT** **Rollno: 328**

OUTPUT -



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\HP\Documents\DS\PRACTICAL8.py ===============
Without any order
12
10
5
Now ordering with insert
4
13
28
123
130
Pre order
28
4
13
130
123
In Order
4
13
28
123
130
Post Order
13
4
123
130
28
>>> 
```