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:Mr.ShivamVishwakarma____

Roll No: _342______ 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:342**
subject: DataStructure__________

Subject: Data Structures

INDEX

Sr No	Date	Topic	Sign
1	04/09/2020	Implement the following for Array: • 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. • 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: • Perform Stack operations using Array implementation. b. • Implement Tower of Hanoi. • WAP to scan a polynomial using linked list and add two polynomials. • 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: • Write a program to implement the collision technique. • 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.	

Shivam Vishwakarma Roll no:342

SYIT

Practical no:1a

Aim:write a program to store the element in 1-D array and provide an option to Perform the operations like searching,sorting,merging,reversing the elements.

Theory: One Dimensional Arrays

A one-dimensional array is one in which only one subscript specification is needed to specify a particular element of the array.

A one-dimensional array is a list of related variables. Such lists are common in programming.Storing Data in Arrays. Assigning values to an elementin an array is similar to assigning
values to scalar variables. Simply reference an individual element of anarray using
the array name and the index inside parentheses, then use the assignment operator (=)
followed by a value.
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.
- Deletion – Deletes an element at the given index.
- Search – Searches an element using the given index or by the value

Program:

```
arr1=[12,35,42,22,1,6,54]
arr2=['hello','world']
arr1.index(35)
print(arr1)
arr1.sort()
print(arr1)
arr1.extend(arr2)
print(arr1)
arr1.reverse()
print(arr1)
```

Output:

Shivam Vishwakarma Roll no:342
SYIT

Practical no:1b

Aim: write a program to perform the matrix addition,multiplication

And transpose operation.

Theory: • add() – add elements of two matrices.

• subtract() – subtract elements of two matrices.

• divide() – divide elements of two matrices.

• multiply() – multiply elements of two matrices.

• dot() – It performs matrix multiplication, does not element wise multiplication.

• sqrt() – square root of each element of matrix.

• sum(x,axis) – add to all the elements in matrix. Second argument is optional, it is

used when we want to compute the column sum if axis is 0 and row sum if axis is 1.

• "T" – It performs transpose of the specified matrix.

Program:

```
#addition
mat1 = [[1, 2], [3, 4]]
mat2 = [[1, 2], [3, 4]]
mat3 = [[0, 0], [0, 0]]

for i in range(0, 2):
    for j in range(0, 2):
        mat3[i][j] = mat1[i][j] + mat2[i][j]

for i in range(0, 2):
    for j in range(0, 2):
```

```
        print(mat3[i][j], end="")
print()

#multiplication
mat1 = [[10, 9], [8, 6]]
mat2 = [[1, 2], [3, 4]]
mat3 = [[0, 0], [0, 0]]

for i in range(0, 2):
    for j in range(0, 2):
        mat3[i][j] = mat1[i][j] * mat2[i][j]


for i in range(len(mat1)):
    for j in range(len(mat2[0])):
        for k in range(len(mat2)):
            mat3[i][j] =mat3[i][j] + ( mat1[i][k] * mat2[k][j])


for r in mat3:
    print(r)

    # Program to transpose a matrix using a nested loop

X = [[12,7],
    [4 ,5],
    [3 ,8]]

result = [[0,0,0],
         [0,0,0]]

# iterate through rows
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)
```

Output:

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\kradd\OneDrive\Desktop\ppp.py ===============
2468
[47, 74]
[50, 64]
[12, 4, 3]
[7, 5, 8]
>>>
```

Shivam Vishwakarma Roll no:342
SYIT

Practical no:2

Aim: Implement linked list, include option 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.

- Insertion in a Linked list: Inserting element in the linked list involves reassigning the pointers from the existing nodes to the newly inserted node. Depending on whether the new data element is getting inserted at the beginning or at the middle or at the end of the linked list.
- Deleting an Item form a Linked List: We can remove an existing node using the key for that node. In the below program we locate the previous node of the node which is to be deleted.

Then point the next pointer of this node to the next node of the node to be deleted.
• Searching in linked list: Searching 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.
• Reversing a Linked list: To reverse a Linked List recursively we need to divide the Linked
List into two parts: head and remaining. Head points to the first element initially. Remaining
points to the next element from the head. We traverse the Linked List recursively until the
second last element.
• Concatenating Linked lists: Concatenate the two lists by traversing the first list until we reach
it’s a tail node and then point the next of the tail node to the head node of the second list.
Store this concatenated list in the first lis

Program:

```
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(list1.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 = list1.get_tail()
    print(last.element)
    while last.previous:
        if type(last.previous.element) == type(list1.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)
    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("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
    return list1.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)
            if type(value.element) == type(list1.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('element 1')
```

```
list1 = LinkedList()
list1.add_head(l1)
list1.add_tail('element 2')
list1.add_tail('element 3')
list1.add_tail('element 4')
list1.get_head().element.element
list1.add_between_list(2,'element between')
list1.remove_between_list(2)

list2 = LinkedList()
l2 = Node('element 5')
list2.add_head(l2)
list2.add_tail('element 6')
list2.add_tail('element 7')
list2.add_tail('element 8')
list1.merge(list2)
list1.get_previous_node_at(3).element
list1.reverse_display()
list1.search('element 6')

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(list1.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 = list1.get_tail()
    print(last.element)
    while last.previous:
        if type(last.previous.element) == type(list1.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)
    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
```

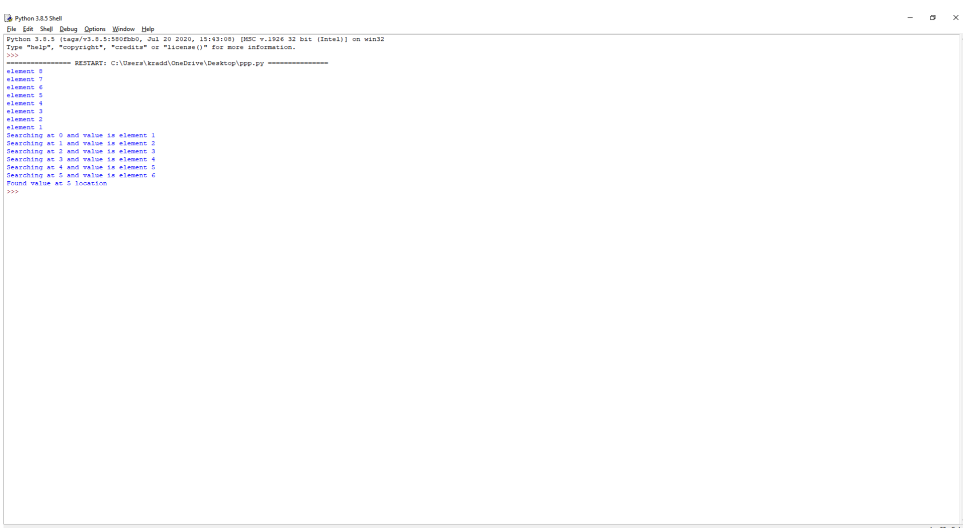
```
    if self.size >= 2:
        first = self.head
        temp_counter = self.size -2
        while temp_counter > 0:
            first = first.next
            temp_counter -= 1
        return first



    else:
        print("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
    return list1.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)
            if type(value.element) == type(list1.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('element 1')
list1 = LinkedList()
list1.add_head(l1)
list1.add_tail('element 2')
list1.add_tail('element 3')
```

```
list1.add_tail('element 4')
list1.get_head().element.element
list1.add_between_list(2,'element between')
list1.remove_between_list(2)

list2 = LinkedList()
l2 = Node('element 5')
list2.add_head(l2)
list2.add_tail('element 6')
list2.add_tail('element 7')
list2.add_tail('element 8')
list1.merge(list2)
list1.get_previous_node_at(3).element
list1.reverse_display()
list1.search('element 6')
```

output:



Shivam Vishwakarma Roll no:342
SYIT

Practical no:3a

Aim: perform stack operation using array implementation

Theory: Stacks is one of the earliest data structures defined in computer science. In simple words,
Stack is a linear collection of items. It is a collection of objectsthat supports fast last-in, first-
out (LIFO) semantics for insertion and deletion. It is an array or list structure of function calls
and parameters used in modern computer programming and CPU architecture. Similar to a stack of plates at a restaurant, elements in a stack are added or removed from the top of the stack, in a "last in, first out" order. Unlike lists or arrays, random access is not allowed for the objects contained in the stack.
There are two types of operations in Stack:

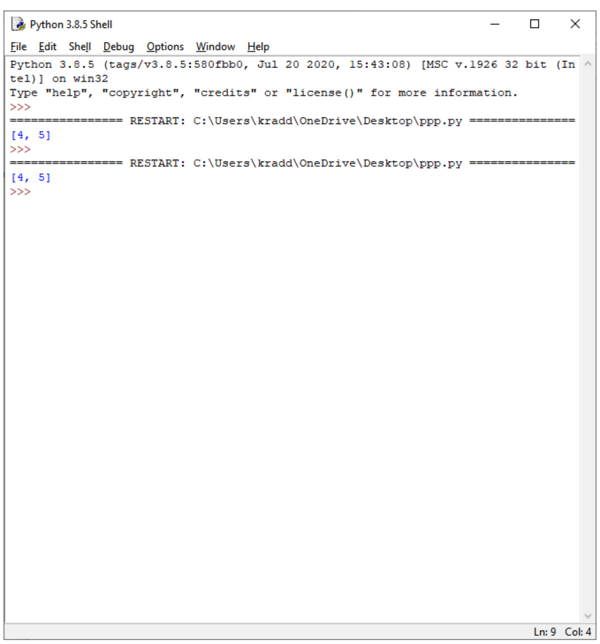
- Push– To add data into the stack.
- Pop– To remove data from the stack

Program:

```
class Stack:
    def __init__(self):
        self.stack_arr = []
```

```
    def push(self,value):
        self.stack_arr.append(value)

    def pop(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            self.stack_arr.pop()

    def get_head(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            return self.stack_arr[-1]

    def display(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            print(self.stack_arr)

stack = Stack()
stack.push(4)
stack.push(5)
stack.push(6)
stack.pop()
stack.display()
stack.get_head()
```

output:


Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
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\kradd\OneDrive\Desktop\ppp.py ===============
[4, 5]
>>>
================ RESTART: C:\Users\kradd\OneDrive\Desktop\ppp.py ===============
[4, 5]
>>>
Ln: 9 Col: 4

Shivam Vishwakarma Roll no:342
SYIT

Practical no:3b

Aim: Implement Tower of Hanoi.

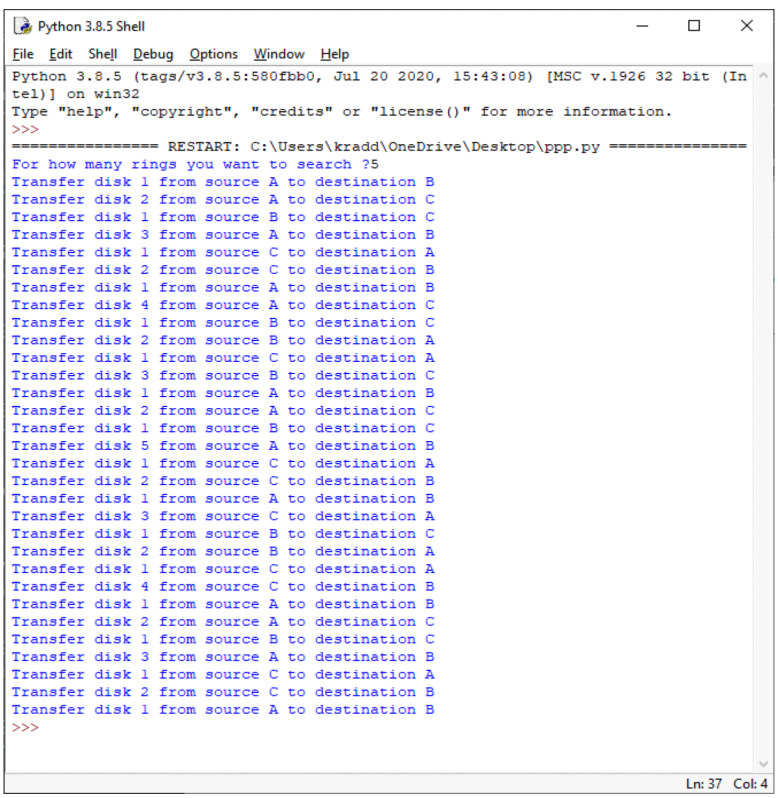
Theory: Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

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.

Program:

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

disk = int(input("For how many rings you want to search ?"))
Tower_of_Hanoi(disk,'A','B','C')
```

Output:



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
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\kradd\OneDrive\Desktop\ppp.py ===============
For how many rings you want to search ?5
Transfer disk 1 from source A to destination B
Transfer disk 2 from source A to destination C
Transfer disk 1 from source B to destination C
Transfer disk 3 from source A to destination B
Transfer disk 1 from source C to destination A
Transfer disk 2 from source C to destination B
Transfer disk 1 from source A to destination B
Transfer disk 4 from source A to destination C
Transfer disk 1 from source B to destination C
Transfer disk 2 from source B to destination A
Transfer disk 1 from source C to destination A
Transfer disk 3 from source B to destination C
Transfer disk 1 from source A to destination B
Transfer disk 2 from source A to destination C
Transfer disk 1 from source B to destination C
Transfer disk 5 from source A to destination B
Transfer disk 1 from source C to destination A
Transfer disk 2 from source C to destination B
Transfer disk 1 from source A to destination B
Transfer disk 3 from source C to destination A
Transfer disk 1 from source B to destination C
Transfer disk 2 from source B to destination A
Transfer disk 1 from source C to destination A
Transfer disk 4 from source C to destination B
Transfer disk 1 from source A to destination B
Transfer disk 2 from source A to destination C
Transfer disk 1 from source B to destination C
Transfer disk 3 from source A to destination B
Transfer disk 1 from source C to destination A
Transfer disk 2 from source C to destination B
Transfer disk 1 from source A to destination B
>>>
Ln: 37 Col: 4
```

Shivam Vishwakarma Roll no:342
SYIT

Practical no:3c

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

Theory: Polynomial is a mathematical expression that consists of variables and coefficients. for example x^2 - 4x +7. 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 a linked list that is used to store Polynomial looks like −Polynomial : 4x7 + 12x2 + 45

Program:

```
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()
    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)
    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("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
        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)
        if value.element == search_value:
            return value.element
        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



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

my_list2 = LinkedList()
my_list2.add_head(Node(int(input(f"Enter coefficient for power
{order} : "))))
for i in reversed(range(order)):
    my_list2.add_tail(int(input(f"Enter coefficient for power {i} : ")))

for i in range(order + 1):
    print(my_list.get_node_at(i).element +
my_list2.get_node_at(i).element)
```

Output:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

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

Ln: 17 Col: 0

Shivam Vishwakarma Roll no:342
SYIT

Practical no:3d

Aim:WAP to calculate factorial and to computer the factors of a given no i)using recursion ii)using iteration.

Theory: The factorial of a number is the product of all the integers from 1 to that number. For
example, the factorial of 6 is 1*2*3*4*5*6 = 720. Factorial is not defined for negative
numbers and the factorial of zero is one, 0! = 1.

• Recursion: 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.

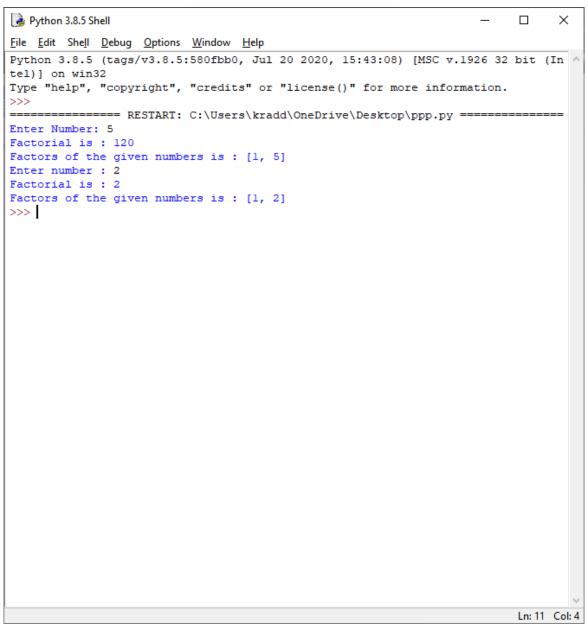
• Iteration: Repeating identical or similar tasks without making errors is something
that computers do well and people do poorly. Repeated execution of a set of
statements is called iteration. Because iteration is so common, Python provides
several language features to make it easier.

Program:

```
factorial = 1
n = int(input('Enter Number: '))
for i in range(1,n+1):
    factorial = factorial * i
```

```
print(f'Factorial is : {factorial}')

fact = []
for i in range(1,n+1):
    if (n/i).is_integer():
        fact.append(i)

print(f'Factors of the given numbers is : {fact}')

factorial = 1
index = 1
n = int(input("Enter number : "))
def calculate_factorial(n,factorial,index):
    if index == n:
        print(f'Factorial is : {factorial}')
        return True
    else:
        index = index + 1
        calculate_factorial(n,factorial * index,index)
calculate_factorial(n,factorial,index)

fact = []
def calculate_factors(n,factors,index):
    if index == n+1:
        print(f'Factors of the given numbers is : {factors}')
        return True
    elif (n/index).is_integer():
        factors.append(index)
        index += 1
        calculate_factors(n,factors,index)
    else:
        index += 1
        calculate_factors(n,factors,index)

index = 1
factors = []
calculate_factors(n,factors,index)
```

output:



```
Python 3.8.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
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\kradd\OneDrive\Desktop\ppp.py ===============
Enter Number: 5
Factorial is : 120
Factors of the given numbers is : [1, 5]
Enter number : 2
Factorial is : 2
Factors of the given numbers is : [1, 2]
>>>
Ln: 11  Col: 4
```

Shivam Vishwakarma Roll no:342
SYIT

Practical no:4

Aim: Perform Queues operation using circular array implementation.

Theory: Circular queue avoids the wastage of space in a regular queue implementation using arrays.

Circular Queue works by the process of circular increment i.e. when we try to increment the
pointer and we reach the end of the queue, we start from the beginning of the queue. Here,
the circular increment is performed by modulo division with the queue size. That is, if REAR
+ 1 == 5 (overflow!), REAR = (REAR + 1)%5 = 0 (start of queue) The circular queue work as
follows:
two pointers FRONT and REAR FRONT track the first element of the queue
REAR track the last elements of the queue initially, set value of FRONT and REARto -1
1. Enqueue Operation check if the queue is full for the first element, set value of FRONT to 0
circularly increase the REAR index by 1 (i.e. if the rear reaches the end, next it would be at
the start of the queue) add the new element in the position pointed to by REAR
2. Dequeue Operation check if the queue is empty return the value pointed by FRONT
circularly increase the FRONT index by 1 for the last element, reset the values

of FRONT and REAR to -1

Program:

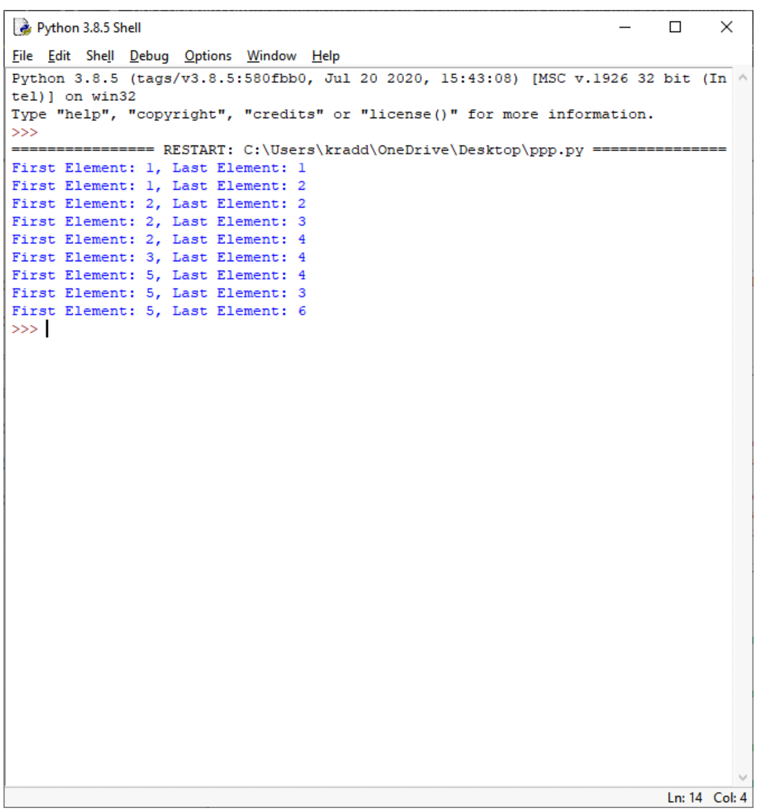
```
class ArrayQueue:
    """FIFO queue implementation using a Python list as underlying
storage."""
    DEFAULT_CAPACITY = 10        # moderate capacity for all new
queues

    def __init__(self):
        """Create an empty queue."""
        self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0
        self._back = 0

    def __len__(self):
        """Return the number of elements in the queue."""
        return self._size

    def is_empty(self):
        """Return True if the queue is empty."""
        return self._size == 0

    def first(self):
        """Return (but do not remove) the element at the front of the
queue.
        Raise Empty exception if the queue is empty.
        """
        if self.is_empty():
            raise Empty('Queue is empty')
        return self._data[self._front]


    def dequeueStart(self):
```

```
        """Remove and return the first element of the queue (i.e.,
FIFO).
        Raise Empty exception if the queue is empty.
        """
        if self.is_empty():
            raise Empty('Queue is empty')
        answer = self._data[self._front]
        self._data[self._front] = None        # help garbage collection
        self._front = (self._front + 1) % len(self._data)
        self._size -= 1
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def dequeueEnd(self):
        """Remove and return the Last element of the queue.
        Raise Empty exception if the queue is empty.
        """
        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        # help garbage collection
        self._front = self._front
        self._size -= 1
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def enqueueEnd(self, e):
        """Add an element to the back of queue."""
        if self._size == len(self._data):
            self._resize(2 * len(self.data))     # double the array size
        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 enqueueStart(self, e):
```

```
        """Add an element to the start of queue."""
        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):                  # we assume cap >= len(self)
        """Resize to a new list of capacity >= len(self)."""
        old = self._data                       # keep track of existing list
        self._data = [None] * cap              # allocate list with new
capacity
        walk = self._front
        for k in range(self._size):            # only consider existing
elements
            self._data[k] = old[walk]           # intentionally shift indices
            walk = (1 + walk) % len(old)        # use old size as modulus
        self._front = 0                        # front has been realigned
        self._back = (self._front + self._size - 1) % len(self._data)

queue = ArrayQueue()
queue.enqueueEnd(1)
print(f"First Element: {queue._data[queue._front]}, Last Element:
{queue._data[queue._back]}")
queue._data
queue.enqueueEnd(2)
print(f"First Element: {queue._data[queue._front]}, Last Element:
{queue._data[queue._back]}")
queue._data
queue.dequeueStart()
print(f"First Element: {queue._data[queue._front]}, Last Element:
{queue._data[queue._back]}")
queue.enqueueEnd(3)
print(f"First Element: {queue._data[queue._front]}, Last Element:
{queue._data[queue._back]}")
```

```
queue.enqueueEnd(4)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.dequeueStart()
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.enqueueStart(5)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.dequeueEnd()
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.enqueueEnd(6)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
```

Output:



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\kradd\OneDrive\Desktop\ppp.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
First Element: 5, Last Element: 6
>>>
```

Shivam Vishwakarma Roll no:342
SYIT

Practical no:5

Aim: write a program to search an element from a list.Give user the option to perform liner or binary search.

Theory: • Linear Search: This linear search is a basic search algorithm which searches all the elements in the list and finds the required value. This is also known as sequential search.

• Binary Search: In computer science, a binary searcher half-interval search algorithm finds the position of a target value within a sorted array. The binary search algorithm can be classified as a dichotomies divide-and-conquer search algorithm and executes in logarithmic time.

Program:

```
print ("* BINARY SEARCH METHOD\n")
def bsm(arr,start,end,num):
    if end>=start:
        mid=start+(end-start)//2
        if arr[mid]==x:
            return mid
        elif arr[mid]>x:
            return bsm(arr,start,mid-1,x)
        else:
            return bsm(arr,mid+1,end,x)
    else:
        return -1
arr=[10,27,36,49,58,69,70]
x=int(input("Enter the number to be searched : "))
result=bsm(arr,0,len(arr)-1,x)
if result != -1:
    print ("Number is found at ",result)
else:
    print ("Number is not present")
```

Output:

Shivam Vishwakarma Roll no:342
SYIT

Practical no:6

Aim: write a program to sort a list of element.Give user the option to perform sorting using insertion sort,bubble sort or selection sort.

Theory: Bubble Sort: Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

• 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 sub arrays in a given array

• Insertion Sort: Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Program:

```
#selection sort
def selection_sort(num):
        for i in range(len(num)):
            lowest_value_index=i
            for j in range(i+1,len(num)):
              if num[j]<num[lowest_value_index]:
                        lowest_value_index=j

num[i],num[lowest_value_index]=num[lowest_value_index],num[i]




list=[1,2,3,4]
selection_sort(list)
print(list)

#insertion sort
def insertionSort(arr):
   for i in range(1, len(arr)):
      key = arr[i]
      j = i-1
      while j >=0 and key < arr[j] :
         arr[j+1] = arr[j]
```

```
        j -= 1
      arr[j+1] = key
# main
arr = ['t','u','t','o','r','i','a','l']
insertionSort(arr)
print ("The sorted array is:")
for i in range(len(arr)):
   print (arr[i])

#bubble sort
def bubble_sort(num):
     swap=True
     while swap:
            swap=False
            for i in range(len(num)-1):
                 if num[i]>num[i+1]:
                       num[i],num[i+1]=num[i+1],num[i]
                       swap=True


list=[23,14,66,8,2]
bubble_sort(list)
print(list)
```

Output:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\kradd\OneDrive\Desktop\ppp.py ===============
[1, 2, 3, 4]
The sorted array is:
a
i
l
o
r
t
t
u
[2, 8, 14, 23, 66]
>>>
```

Ln: 16 Col: 4

Shivam Vishwakarma Roll no:342
SYIT

Practical no:7a

Aim:write a program to implement the collision technique.

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.
• Collisions: A Hash Collision Attack is an attempt to find two input strings of
a hash function that produce the same hash result. If two separate inputs produce
the same hash output, it is called a collision.
• Collision Techniques: 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
• Separate Chaining: The idea is to make each cell of hash table point to a linked list of
records that have same hash function value.
• Open Addressing: Like separate chaining, open addressing is a method for handling
collisions. In Open Addressing, all elements are stored in the hash table itself. So at

any point, the size of the table must be greater than or equal to the total number of
keys (Note that we can increase table size by copying old data if needed)

Program:

```
class Hash:
    def _init_(self, keys: int, lower_range: int, higher_range: int) ->
None:
        self.value = self.hash_function(keys, lower_range,
higher_range)

    def get_key_value(self) -> int:
        return self.value

    @staticmethod
    def hash_function(keys: int, lower_range: int, higher_range: int) -
> int:
        if lower_range == 0 and higher_range > 0:
            return keys % higher_range


if _name_ == '_main_':
    linear_probing = True
    list_of_keys = [23, 43, 1, 87]
    list_of_list_index = [None]*4
    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()
        print("Hash value for " + str(value) + " is :" + str(list_index))
        if list_of_list_index[list_index]:
            print("Collision 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))
```

Output:

Shivam Vishwakarma Roll no:342
SYIT

Practical no:7b

Aim: write a program to implement the concept of liner 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. Along with quadratic probing and double hashing,linear
probing is a form of open addressing.

Program:
```
size_list = 6

def hash_function(val):
    global size_list
    return val%size_list

def map_hash_function(hash_return_values):
    return hash_return_values

def create_hash_table(list_values,main_list):
```

```
    for  values in list_values:
        hash_return_values = hash_function(values)
        list_index = map_hash_function(hash_return_values)
        if main_list[list_index]:
            print("collision detected")
            linear_probing(list_index,values,main_list)
        else:
            main_list[list_index]=values

def linear_probing(list_index,value,main_list):
    global size_list
    list_full = False
    old_list_index=list_index
    if list_index == size_list - 1:
        list_index = 0
    else:
        list_index += 1

    while main_list[list_index]:
        if list_index+1 == size_list:
            list_index = 0
        else:
            list_index += 1
        if list_index == old_list_index:
            list_full = True
            break
    if list_full == True:
         print("list was full. could not saved")




def search_list(key,main_list):
    #for i in range(size_list):
```

```
    val = hash_function(key)
    if main_list[val] == key:
        print("list found",val)
    else:
        print("not found")


list_values = [1,3,8,6,5,14]

main_list = [None for x in range(size_list)]
print(main_list)
create_hash_table(list_values,main_list)
print(main_list)
search_list(5,main_list)
```

Output:

Shivam Vishwakarma Roll no:342
SYIT

Practical no:8

Aim:write a program for inorder postorder and preorder traversal of tree.

Theory: 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.
• 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.
• Postorder: Postorder traversal is also useful to get the postfix expression of an
expression tree.

Program:

```
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:
                    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()
```

```python
print("Post Order")
root_1.Printpostorder()
```

output: