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. **Jainab Bee Kalim Shah**

Roll No: **335** 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: 335**

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.	

PRACTICAL NO.01

Implement the following for Array:

A) 1-D Array (Searching, Sorting, Merging & Reversing)

AIM –

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

THEORY –

1-D Array - 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. To create a one dimensional array in Numpy, you can use either of the array(), arange() or linspace() numpy functions.

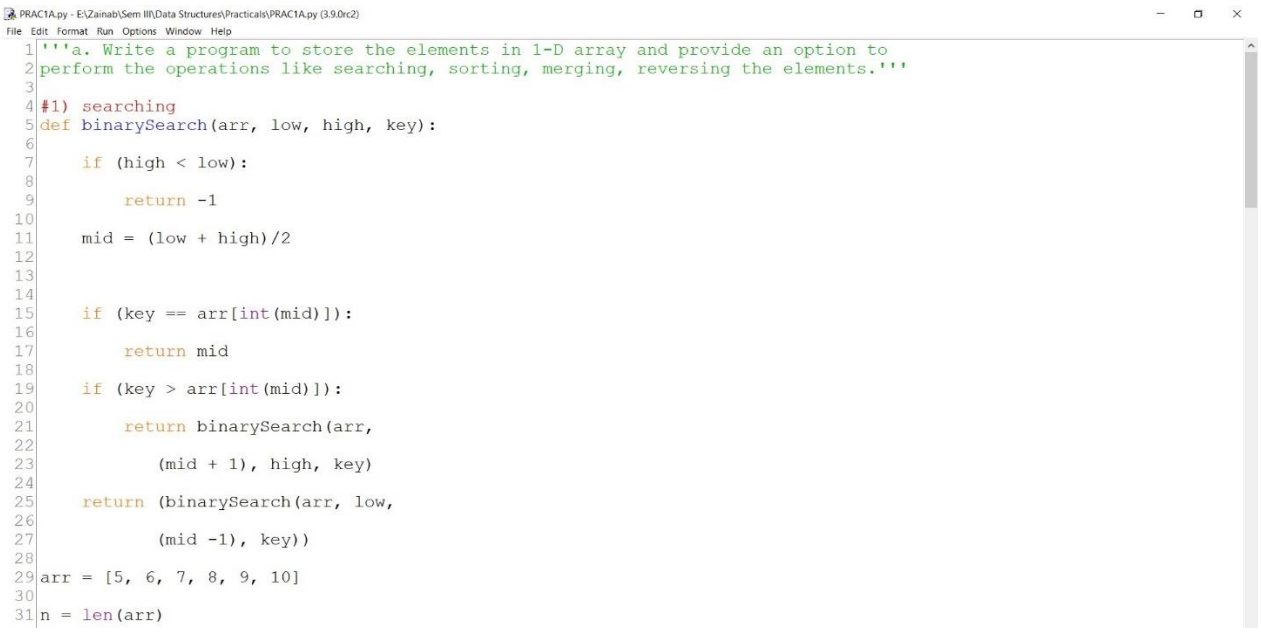
Reversing - To reverse an array using an additional array, using swapping and by using a function.

Merging - To merge array elements we have to copy first array's elements into third array first then copy second array's elements into third array after the index of first array elements.

Sorting - Returns a sorted version of array with the elements arranged in ascending order. If array is an array of clusters, the function sorts the elements by comparing the first elements. If the first elements match, the function compares the second and subsequent elements. The connector pane displays the default data types for this polymorphic function.

Searching - The Search 1D Array function returns the index of the first occurrence of an element in a one-dimensional array or -1 if no match is found. Search starts at the specified start index or at index zero if no start index is specified.

CODE –




```
'''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.'''

#1) searching
def binarySearch(arr, low, high, key):

    if (high < low):

        return -1

    mid = (low + high)/2


    if (key == arr[int(mid)]):

        return mid

    if (key > arr[int(mid)]):

        return binarySearch(arr,

           (mid + 1), high, key)

    return (binarySearch(arr, low,

           (mid -1), key))

arr = [5, 6, 7, 8, 9, 10]

n = len(arr)
```

PRAC1A.py - E:\Zainab\Sem III\Data Structures\Practicals\PRAC1A.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
key = 10

print("Index:", int(binarySearch(arr, 0, n, key) ))


#2) Sorting

numbers = [1, 3, 4, 2]
# Sorting list of Integers
numbers.sort()
print(numbers)

#3) Merging
def mergeArrays(arr1, arr2, n1, n2):

    arr3 = [None] * (n1 + n2)

    i = 0

    j = 0

    k =0

    while i < n1 and j < n2:

        if arr1[i] < arr2[j]:

            arr3[k] = arr1[i]

            k = k + 1
```

PRAC1A.py - E:\Zainab\Sem III\Data Structures\Practicals\PRAC1A.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
            i = i + 1

        else:

            arr3[k] = arr2[j]

            k = k + 1

            j = j + 1

    while i < n1:

        arr3[k] = arr1[i];

        k = k + 1

        i = i +1

    while j < n2:

        arr3[k] = arr2[j];

        k = k + 1

        j = j + 1

    print("Array After Merging")

    for i in range(n1 + n2):
```

GitHub URL: https://github.com/imszainab/DSPracticals

PRAC1A.py - E:\Zainab\Sem III\Data Structures\Practicals\PRAC1A.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
        print(str(arr3[i]), end = " ")

arr1 = [1, 3, 5, 7]

n1 = len(arr1)
arr2 = [2, 4, 6, 8]

n2 = len(arr2)
mergeArrays(arr1, arr2, n1, n2);

#4) reverse
def reverseList(A, start, end):

    while start < end:

        A[start], A[end] = A[end], A[start]

        start += 1

        end -=1

A = [1, 2, 3, 4, 5, 6]

print(A)

reverseList(A, 0, 5)

print("Reversed List is")

print(A)
```

OUT PUT –

Python 3.9.0rc2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: E:\Zainab\Sem III\Data Structures\Practicals\PRAC1A.py =======
Index: 5
[1, 2, 3, 4]
Array After Merging
1 2 3 4 5 6 7 8 [1, 2, 3, 4, 5, 6]
Reversed List is
[6, 5, 4, 3, 2, 1]
>>>
```

B) Matrix Operations (Addition, Multiplication and Transpose)

AIM **-** Write a program to perform the Matrix addition, Multiplication and Transpose

Operation.

Matrix Multiplication **-** Matrix multiplication is a binary operation that produces a matrix from two matrices. For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix.

Matrix Addition **-** Addition is the operation of adding two matrices by adding the corresponding entries together. However, there are other operations which could also be considered addition for matrices, such as the direct sum and the Kronecker sum.

GitHub URL: https://github.com/imszainab/DSPracticals

Transpose - In linear algebra, the transpose of a matrix is an operator which flips a matrix over its diagonal; that is, it switches the row and column indices of the matrix A by producing another matrix, often denoted by A^T.

CODE –

PRAC1B.py - E:\Zainab\Sem III\Data Structures\Practicals\PRAC1B.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
'''Practical 1 (b)Write a program to perform the Matrix addition, Multiplication and Transpose
Operation.'''
Matrix1 = [[3, 4, -6],
      [12,71,24],
      [21,3,21]]

Matrix2 = [[2, 16, -16],
           [1,7,-3],
           [-1,3,3]]
Matrix3 = [[0,0,0,],
     [0,0,0,],
     [0,0,0,]]

# Matrix Addition
for i in range(len(Matrix1)):
    for j in range(len(Matrix2[0])):
        for k in range(len(Matrix2)):
            Matrix3[i][j] += Matrix1[i][k] + Matrix2[k][j]

print(Matrix3)

# Matrix Multiplication

Matrix3 = [[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]]

for i in range(len(Matrix1)):
    for j in range(len(Matrix2[0])):
        for k in range(len(Matrix2)):
            Matrix3[i][j] += Matrix1[i][k] * Matrix2[k][j]


print(Matrix3)

#matrix transpose
for i in map(list, zip(*Matrix1)):
    print(i)

```

OUTPUT –

Python 3.9.0rc2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: E:\Zainab\Sem III\Data Structures\Practicals\PRAC1B.py =======
[[3, 27, -15], [109, 133, 91], [47, 71, 29]]
[[16, 58, -78, 0], [71, 761, -333, 0], [24, 420, -282, 0]]
[3, 12, 21]
[4, 71, 3]
[-6, 24, 21]
>>>
```

GitHub URL: https://github.com/imszainab/DSPracticals

PRACTICAL NO.02 -

AIM –

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

THEORY –

A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library.

1. Singly Linked List - In its most basic form, a linked list is a string of nodes, sort of like a string of pearls, with each node containing both data and a reference to the next node in the list (Note: This is a singly linked list. The nodes in a doubly linked list will contain references to both the next node and the previous node). The main advantage of using a linked list over a similar data structure, like the static array, is the linked list's dynamic memory allocation: if you don't know the amount of data you want to store beforehand, the linked list can adjust on the fly.* Of course this advantage comes at a price: dynamic memory allocation requires more space and commands slower look up times.

2. Doubly Linked List - Doubly Linked List contains a link element called first and last. Each link carries a data field(s) and two link fields called next and prev. Each link is linked with its next link using its next link. Each link is linked with its previous link using its previous link.

CODE –


```
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):
```

GitHub URL: https://github.com/imszainab/DSPracticals

Practical 2.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 2.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
                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):
        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

```

Practical 2.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 2.py (3.9.0rc2)

File Edit Format Run Options Window Help

```

    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
            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")
```

GitHub URL: https://github.com/imszainab/DSPracticals

Practical 2.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 2.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
        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")
```

Practical 2.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 2.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
        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):
```

GitHub URL: https://github.com/imszainab/DSPracticals

Practical 2.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 2.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
            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('Jainab')
my_list = LinkedList()
my_list.add_head(l1)
my_list.add_tail('Ayesha')
my_list.add_tail('Amreen')
my_list.add_tail('Sameer')
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('Rahim')
my_list2.add_head(l2)
my_list2.add_tail('Sumaiyya')
my_list2.add_tail('Anabiya')
my_list2.add_tail('Ali')
my_list.merge(my_list2)
my_list.get_previous_node_at(3).element
my_list.reverse_display()
```

OUTPUT –

Python 3.9.0rc2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:\Zainab\Sem III\Data Structures\Practicals\Practical 2.py =====
Ali
Anabiya
Sumaiyya
Rahim
Sameer
Amreen
Ayesha
Jainab
>>>
```

GitHub URL: https://github.com/imszainab/DSPracticals

PRACTICAL NO.03

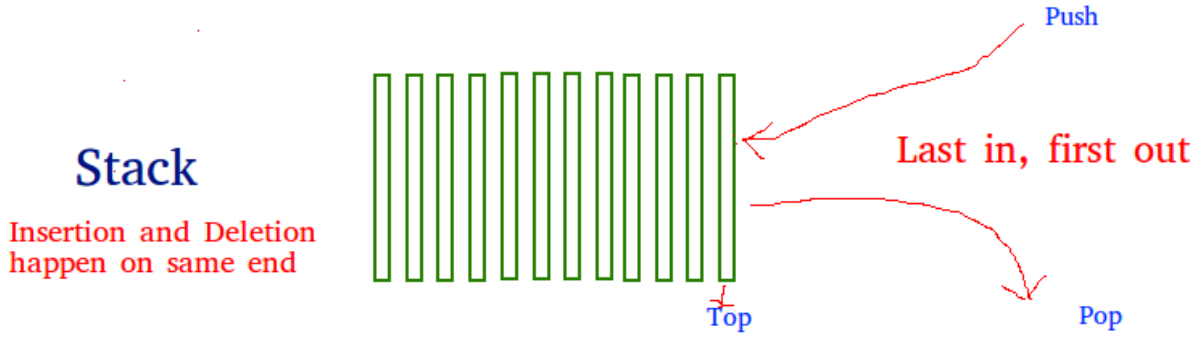
Implement The Following For Stack:

A) Stack operations (Array Implementation.)

AIM – Perform Stack Operations Using Array Implementation.

THEORY -

Stack in Python - A stack is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



The functions associated with stack are:

- empty() – Returns whether the stack is empty – Time Complexity : O(1)
- size() – Returns the size of the stack – Time Complexity : O(1)
- top() – Returns a reference to the top most element of the stack – Time Complexity : O(1)
- push(g) – Adds the element ‘g’ at the top of the stack – Time Complexity : O(1)
- pop() – Deletes the top most element of the stack – Time Complexity : O(1)

Implementation -

There are various ways from which a stack can be implemented in Python. This article covers the implementation of stack using data structures and modules from Python library.

Stack in Python can be implemented using following ways:

- List
- Collections.deque
- Queue.LifoQueue

1. Implementation Using List:

Python’s built-in data structure list can be used as a stack. Instead of push(), append() is used to add elements to the top of stack while pop() removes the element in LIFO order.

GitHub URL: https://github.com/imszainab/DSPracticals

Unfortunately, list has a few shortcomings. The biggest issue is that it can run into speed issue as it grows. The items in list are stored next to each other in memory, if the stack grows bigger than the block of memory that currently hold it, then Python needs to do some memory allocations. This can lead to some append() calls taking much longer than other ones.

2. Implementation Using Collections.Deque:

Python stack can be implemented using deque class from collections module. Deque is preferred over list in the cases where we need quicker append and pop operations from both the ends of the container, as deque provides an O(1) time complexity for append and pop operations as compared to list which provides O(n) time complexity.

The same methods on deque as seen in list are used, append() and pop().

3. Implementation Using Queue Module

Queue module also has a LIFO Queue, which is basically a Stack. Data is inserted into Queue using put() function and get() takes data out from the Queue.

There are various functions available in this module:

- maxsize – Number of items allowed in the queue.
- empty() – Return True if the queue is empty, False otherwise.
- full() – Return True if there are maxsize items in the queue. If the queue was initialized with maxsize=0 (the default), then full() never returns True.
- get() – Remove and return an item from the queue. If queue is empty, wait until an item is available.
- get_nowait() – Return an item if one is immediately available, else raise QueueEmpty.
- put_nowait(item) – Put an item into the queue without blocking.

CODE –

GitHub URL: https://github.com/imszainab/DSPracticals

```
stack.push(5)
stack.push(6)
stack.pop()
stack.display()
stack.get_head()
```

OUTPUT –

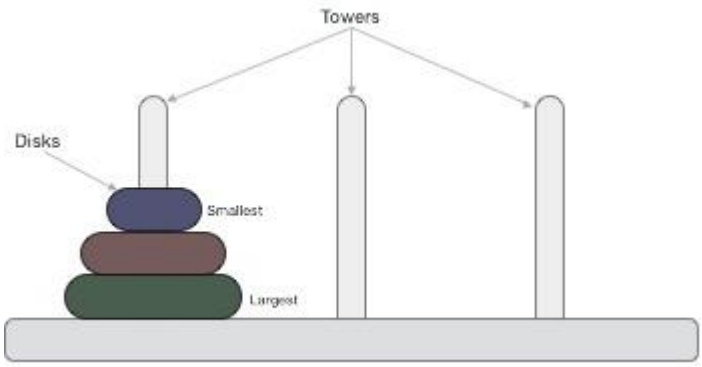
```
Python 3.9.0rc2 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 A.py ======
[4, 5]
>>>
```

B) Tower Of Hanoi

AIM – Implement Tower of Hanoi

THEORY –

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted −



These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Rules -

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

Only one disk can be moved among the towers at any given time.

Only the "top" disk can be removed.

No large disk can sit over a small disk.

GitHub URL: https://github.com/imszainab/DSPracticals

CODE –

Practical 3 B.py - C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 B.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
def hanoi(n, P1, P2, P3):
    """ Move n discs from pole P1 to pole P3. """
    if n == 0:
        # No more discs to move in this step
        return

    global count
    count += 1

    # move n-1 discs from P1 to P2
    hanoi(n-1, P1, P3, P2)

    if P1:
        # move disc from P1 to P3
        P3.append(P1.pop())
        print(A, B, C)

    # move n-1 discs from P2 to P3
    hanoi(n-1, P2, P1, P3)

# Initialize the poles: all n discs are on pole A.
n = 3
A = list(range(n,0,-1))
B, C = [], []

print(A, B, C)
count = 0
hanoi(n, A, B, C)
print(count)

```

OUTPUT –

Python 3.9.0rc2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 B.py ======
[3, 2, 1] [] []
[3, 2] [] [1]
[3] [2] [1]
[3] [2, 1] []
[] [2, 1] [3]
[1] [2] [3]
[1] [] [3, 2]
[] [] [3, 2, 1]
7
>>>
```

C) Linked List (Polynomial)

AIM **-** WAP To Scan A Polynomial Using Linked List And Add Two Polynomial.

THEORY –

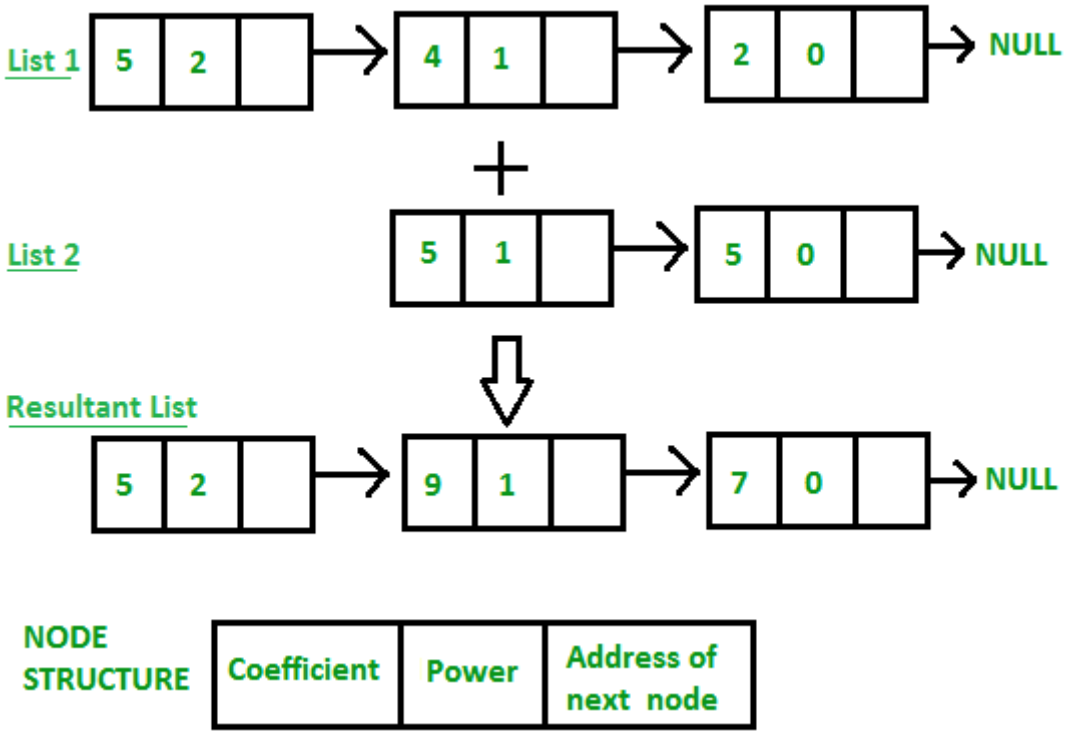
Linked List **-** A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library.

Polynomials **-** Polynomials are the algebraic expressions which consist of variables and coefficients. Variables are also sometimes called indeterminate. We can perform the arithmetic operations such as addition, subtraction, multiplication and also positive integer exponents for polynomial expressions but not division by variable.

GitHub URL: https://github.com/imszainab/DSPracticals

Adding two polynomials using Linked List –

Given two polynomial numbers represented by a linked list. Write a function that add these lists means add the coefficients who have same variable powers.



CODE –

Practical 3 C.py - C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 C.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
#Practical 3 (c) WAP to scan a polynomial using linked list and add two polynomial.
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")
```

GitHub URL: https://github.com/imszainab/DSPracticals

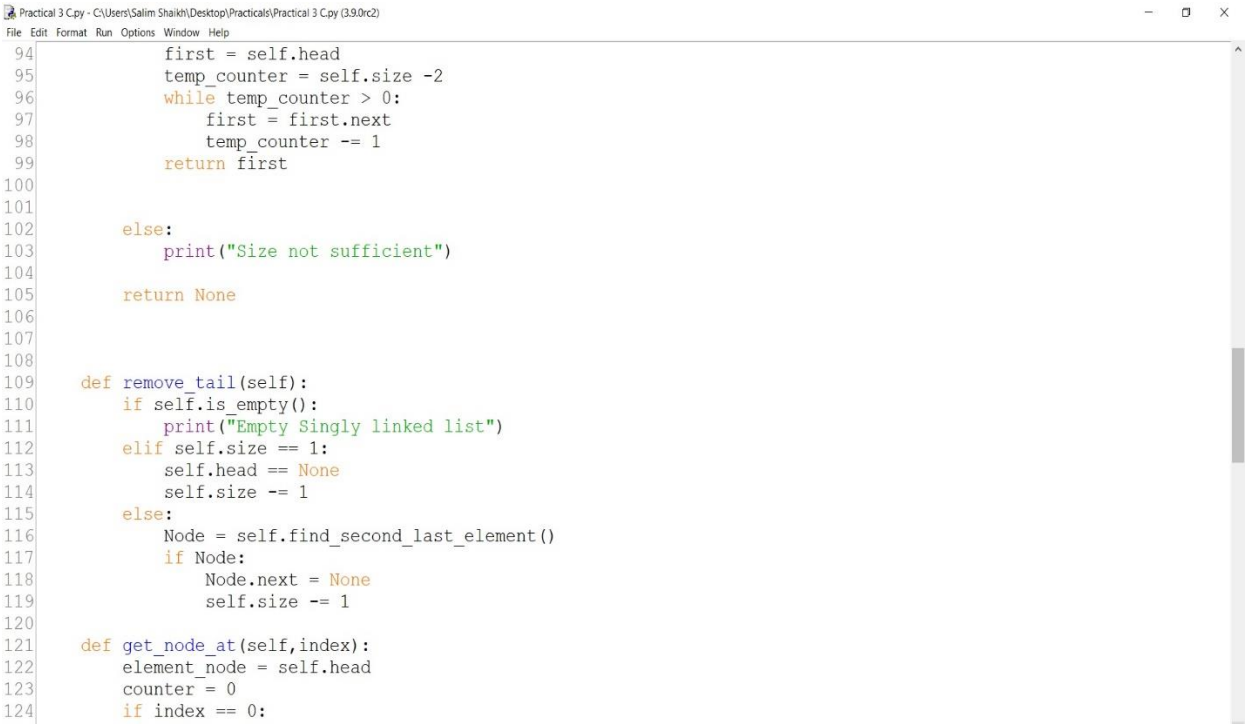
```
            return
        first = self.head
        print(first.element.element)
        first = first.next
        while first:
            if type(first.element) == type(my_list.head.element):
                print(first.element.element)
                first = first.next
            print(first.element)
            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
```

Practical 3 C.py - C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 C.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
        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:
```

GitHub URL: https://github.com/imszainab/DSPracticals




```
            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)
```

GitHub URL: https://github.com/imszainab/DSPracticals

Practical 3 C.py - C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 C.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
        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 3.9.0rc2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 C.py ======
Enter the order for polynomial : 4
Enter coefficient for power 4 : 3
Enter coefficient for power 3 : 3
Enter coefficient for power 2 : 2
Enter coefficient for power 1 : 1
Enter coefficient for power 0 : 0
Enter coefficient for power 4 :
```

GitHub URL: https://github.com/imszainab/DSPracticals

D) Factorial (Recursion & Iteration)

AIM **-** WAP To Calculate Factorial And To Compute The Factors Of A Given No. (I) Using Recursion, (Ii) Using Iteration.

THEORY –

Factorial - 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.

What is recursion?

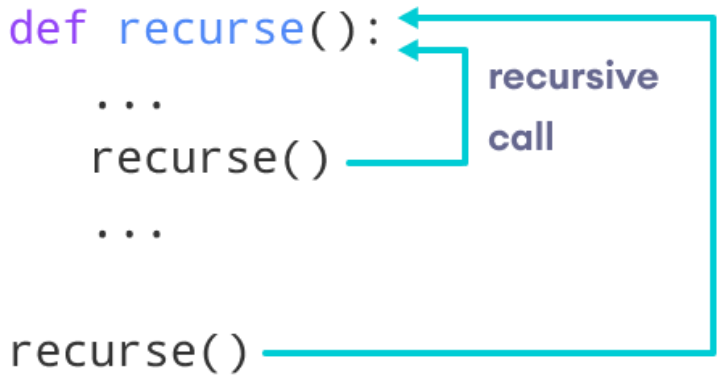
Recursion is the process of defining something in terms of itself.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

Python Recursive Function -

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.

The following image shows the working of a recursive function called recurse.



Iteration (Using For Loop) –

This method finds the factorial of a number by iterating in reverse direction starting from the number till 1 and finding the product in every iteration.

Algorithm for this method comprises of the following steps.

- Initialize a variable to hold the factorial of number to 1.
- Loop from the number till 1 with an interval of 1.
- In every iteration, find the product of number and the variable initialized in step 1 and store it in the same variable

After loop completes (when the last number is 1) the variable will be holding the product of all numbers till 1 which is the factorial of the given number.

GitHub URL: https://github.com/imszainab/DSPracticals

CODE –

Practical 3 D.py - C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 D.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
'''Practical 3(d)WAP to calculate factorial and to compute the factors of a given no. (i) using
recursion, (ii) using iteration'''
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 3.9.0rc2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:\Users\Salim Shaikh\Desktop\Practicals\Practical 3 D.py ======
Enter Number: 8
Factorial is : 40320
Factors of the given numbers is : [1, 2, 4, 8]
Enter number : 14
Factorial is : 87178291200
Factors of the given numbers is : [1, 2, 7, 14]
>>>
```

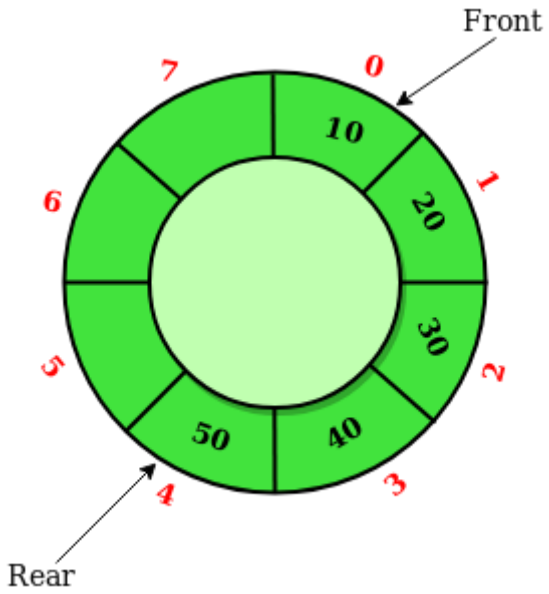
GitHub URL: https://github.com/imszainab/DSPracticals

PRACTICAL NO.04

AIM –

Perform Queues operations using Circular Array implementation.

THEORY - Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called 'Ring Buffer'.



In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we cannot insert the next element even if there is a space in front of queue.

CODE –

Practical 4.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 4.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
class CircularQueue():

        # constructor
        def __init__(self, size): # initializing the class
                self.size = size

                # initializing queue with none
                self.queue = [None for i in range(size)]
                self.front = self.rear = -1

        def enqueue(self, data):

                # condition if queue is full
                if ((self.rear + 1) % self.size == self.front):
                        print(" Queue is Full\n")

                # condition for empty queue
                elif (self.front == -1):
                        self.front = 0
                        self.rear = 0
                        self.queue[self.rear] = data
                else:

                        # next position of rear
                        self.rear = (self.rear + 1) % self.size
                        self.queue[self.rear] = data

        def dequeue(self):
                if (self.front == -1): # codition for empty queue
                        print ("Queue is Empty\n")

```

GitHub URL: https://github.com/imszainab/DSPracticals

```
Practical 4.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 4.py (3.9.0rc2)
File Edit Format Run Options Window Help
                # condition for only one element
                elif (self.front == self.rear):
                        temp=self.queue[self.front]
                        self.front = -1
                        self.rear = -1
                        return temp
                else:
                        temp = self.queue[self.front]
                        self.front = (self.front + 1) % self.size
                        return temp

        def display(self):

                # condition for empty queue
                if(self.front == -1):
                        print ("Queue is Empty")

                elif (self.rear >= self.front):
                        print("Elements in the circular queue are:",
                                                                    end = " ")
                        for i in range(self.front, self.rear + 1):
                                print(self.queue[i], end = " ")
                        print ()

                else:
                        print ("Elements in Circular Queue are:",
                                                                end = " ")
                        for i in range(self.front, self.size):
                                print(self.queue[i], end = " ")
                        for i in range(0, self.rear + 1):
                                print(self.queue[i], end = " ")
```

```
Practical 4.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 4.py (3.9.0rc2)
File Edit Format Run Options Window Help
                        for i in range(self.front, self.rear + 1):
                                print(self.queue[i], end = " ")
                        print ()

                else:
                        print ("Elements in Circular Queue are:",
                                                                end = " ")
                        for i in range(self.front, self.size):
                                print(self.queue[i], end = " ")
                        for i in range(0, self.rear + 1):
                                print(self.queue[i], end = " ")
                        print ()

                if ((self.rear + 1) % self.size == self.front):
                        print("Queue is Full")


ob = CircularQueue(5)
ob.enqueue(14)
ob.enqueue(22)
ob.enqueue(13)
ob.enqueue(-6)
ob.display()
print ("Deleted value = ", ob.dequeue())
print ("Deleted value = ", ob.dequeue())
ob.display()
ob.enqueue(9)
ob.enqueue(20)
ob.enqueue(5)
ob.display()
```

OUTPUT –

```
Python 3.9.0rc2 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:\Zainab\Sem III\Data Structures\Practicals\Practical 4.py =====
Elements in the circular queue are: 14 22 13 -6
Deleted value =  14
Deleted value =  22
Elements in the circular queue are: 13 -6
Elements in Circular Queue are: 13 -6 9 20 5
Queue is Full
>>>
```

GitHub URL: https://github.com/imszainab/DSPracticals

PRACTICAL NO. 05

AIM –

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

THEORY –

1. Linear Search - Linear search is the simplest searching algorithm that searches for an element in a list in sequential order. We start at one end and check every element until the desired element is not found.

2. Binary Search - Binary Search is a searching algorithm for finding an element's position in a sorted array.

In this approach, the element is always searched in the middle of a portion of an array.

Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first.

Binary Search Algorithm can be implemented in two ways which are discussed below.

Iterative Method – The Iterative Method Follows Multiple Iterations To Get Desired Result.

Recursive Method - The recursive method follows the divide and conquer approach.

CODE –

Practical 5.py - C:\Users\Salim Shaikh\Desktop\Practicals\Practical 5.py (3.9.0rc2)

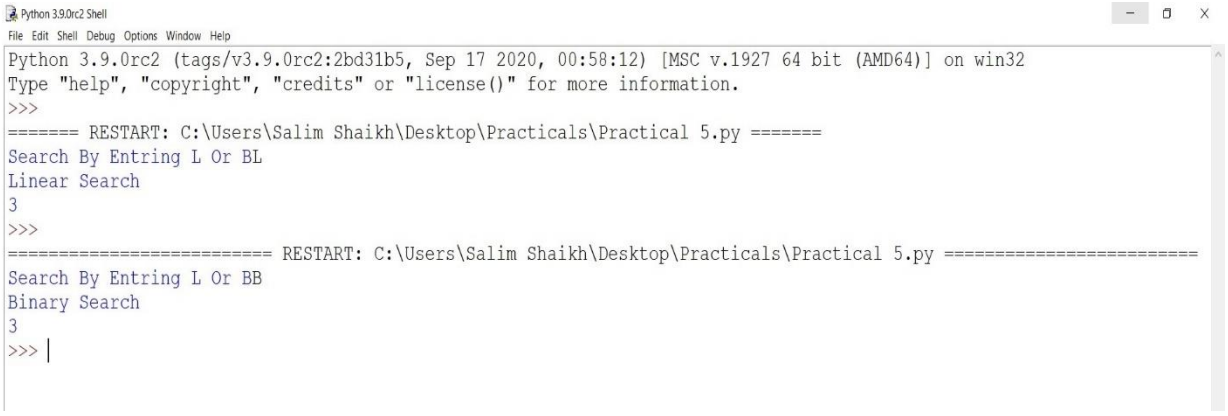
File Edit Format Run Options Window Help

```
def LinearSearch(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i

def BinarySearch(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:
        mid = (high + low) // 2
        if arr[mid] < x:
            low = mid + 1

        elif arr[mid] > x:
            high = mid - 1

        else:
            return mid


arr = [ 2, 3, 4, 10, 40 ]
x = 10

opt = input("Search By Entring L Or B")
if opt == "B":
    result= BinarySearch(arr,x)
    print("Binary Search")
    print(result)
```

GitHub URL: https://github.com/imszainab/DSPracticals

```
32
33 elif opt == "L":
34     result= LinearSearch(arr,x)
35     print("Linear Search")
36     print(result)
37
```

OUTPUT –



GitHub URL: https://github.com/imszainab/DSPracticals

PRACTICAL NO.06

AIM –

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

THEORY –

1. 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.

Algorithm -

To sort an array of size n in ascending order:

1: Iterate from arr[1] to arr[n] over the array.

2: Compare the current element (key) to its predecessor.

3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

2. 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.

3. Bubble Sort - Bubble sort is an algorithm that compares the adjacent elements and swaps their positions if they are not in the intended order. The order can be ascending or descending.

Starting from the first index, compare the first and the second elements. If the first element is greater than the second element, they are swapped.

Now, compare the second and the third elements. Swap them if they are not in order.

The above process goes on until the last element.

CODE -

Practical 6.py - C:\Users\Salim Shaikh\Desktop\Practicals\Practical 6.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
#Code Block To Perform Selection Sort
def SelectionSort(A):
    for i in range(len(A)):
        small = i
        for j in range(i+1, len(A)):
            if A[small] > A[j]:
                small = j
        A[i], A[small] = A[small], A[i]


#Code Block To Perform 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


#Code Block Perform Bubble Sort
def SubbleSort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]


A = [64, 25, 12, 22, 11]
opt = input("Search By Entering I,B or S ")
if opt == 'I' :
    BubbleSort(A)
    print("This Is Insertion Sort")
    print(A)
elif opt == 'B':
    InsertionSort(A)
    print ("This Is Bubble Sort")
    print(A)
elif opt == 'S':
    SelectionSort(A)
    print ("This Is Selection Sort")
    print(A)

```

OUTPUT –

Python 3.9.0rc2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:\Users\Salim Shaikh\Desktop\Practicals\Practical 6.py =======
Search By Entering I,B or S I
This Is Insertion Sort
[11, 12, 22, 25, 64]
>>>
========== RESTART: C:\Users\Salim Shaikh\Desktop\Practicals\Practical 6.py ==========
Search By Entering I,B or S B
This Is Bubble Sort
[11, 12, 22, 25, 64]
>>>
========== RESTART: C:\Users\Salim Shaikh\Desktop\Practicals\Practical 6.py ==========
Search By Entering I,B or S S
This Is Selection Sort
[11, 12, 22, 25, 64]
>>>
```

GitHub URL: https://github.com/imszainab/DSPracticals

PRACTICAL NO.07

Implement the following for Hashing:

A] Collision Technique.

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.

Symbol Tables in Compiler/Interpreter, Dictionaries, caches, etc.

How Hash Function Works?

- It should always map large keys to small keys.
- It should always generate values between 0 to m-1 where m is the size of the hash table.
- It should uniformly distribute large keys into hash table slots.

What is Collision?

Since a hash function gets us a small number for a key which is a big integer or string, there is a possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision and must be handled using some collision handling technique.

CODE –

Practical 7 A.py - E:/Zainab/Sem III/Data Structures/Practicals/Practical 7 A.py (3.9.0rc2)

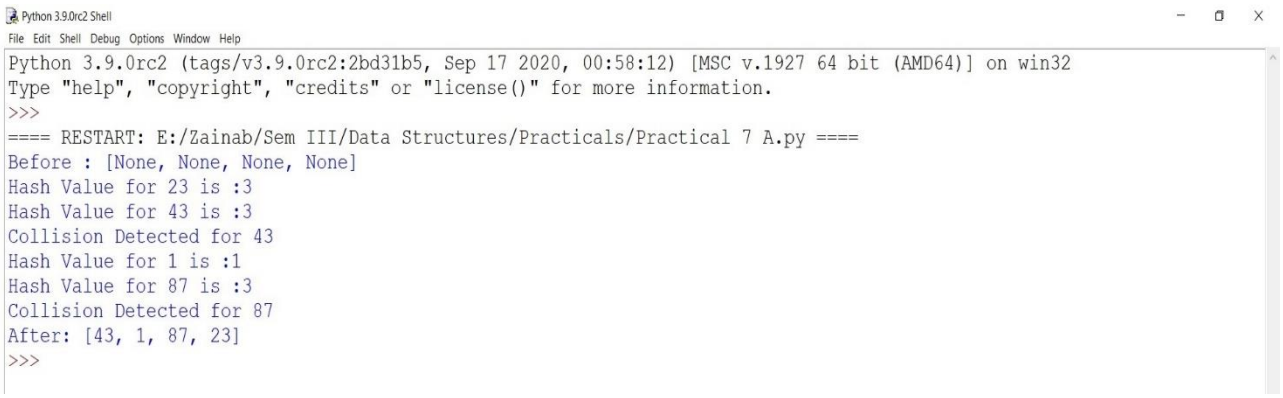
File Edit Format Run Options Window Help

```
#Practical 7a Write a program to implement the collision technique.
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
```

GitHub URL: https://github.com/imszainab/DSPracticals

```
                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 –



B] Linear Probing.

AIM – Write a program to implement the concept of linear probing.

THEORY –

Linear Probing - 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.

When collision occurs, we linearly probe for the next bucket.
We keep probing until an empty bucket is found.

CODE –

Practical 7 B.py - E:/Zainab/Sem III/Data Structures/Practicals/Practical 7 B.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
#Practical 7B - Write a program to implement the concept of linear probing.
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

Python 3.9.0rc2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0rc2 (tags/v3.9.0rc2:2bd31b5, Sep 17 2020, 00:58:12) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: E:/Zainab/Sem III/Data Structures/Practicals/Practical 7 B.py ====
[None, None, None, None, None, None]
Collision Detected
[6, 1, 8, 3, None, 5]
List found 5
>>>
```

GitHub URL: https://github.com/imszainab/DSPracticals

PRACTICAL NO. 08

AIM –

Write A Program for Inorder, Postorder And Preorder Traversal Of Tree.

THEORY –

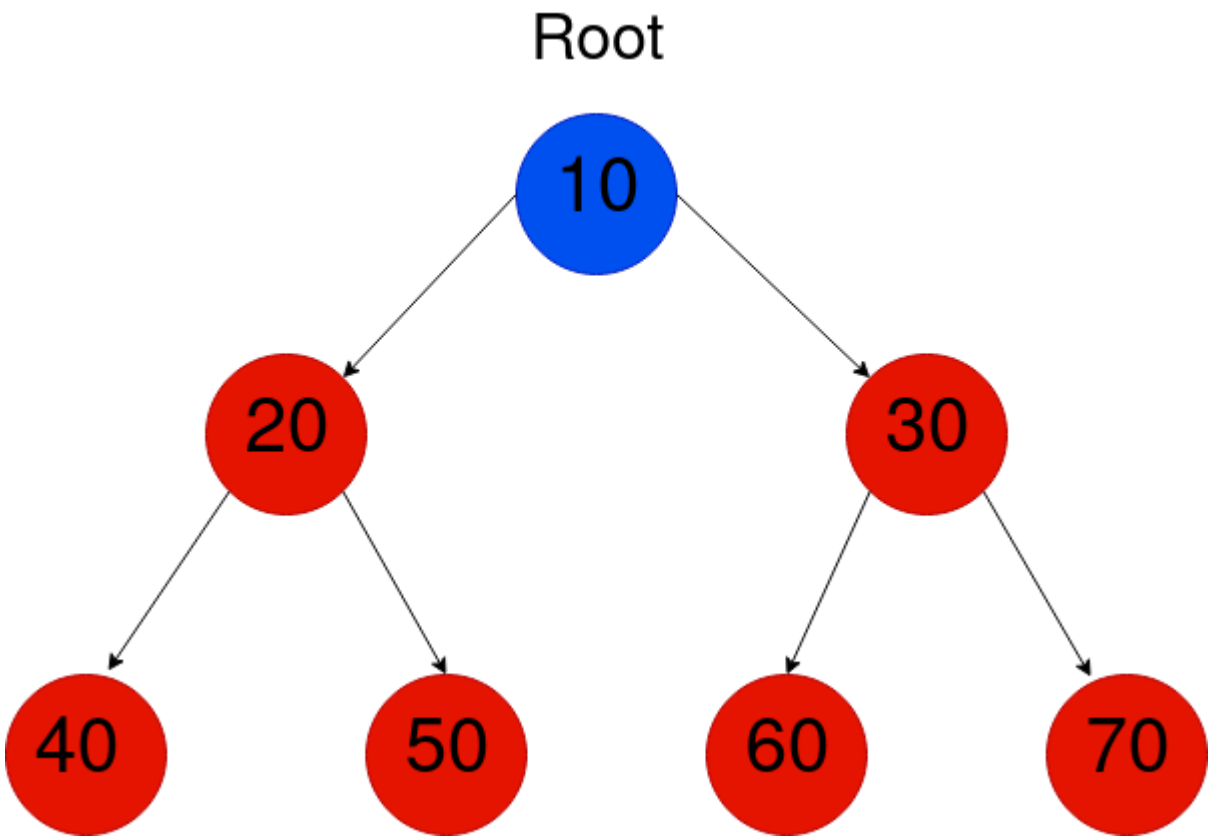
Binary Tree Traversal (PreOrder, InOrder, PostOrder)

A Binary Tree is a data structure where every node has at most two children. We call the topmost node as the Root node.

Since it could have two children, we could move across the Binary Tree in different ways. Here, we will discuss the three most commonly used methods for traversal, namely:

- PreOrder Traversal
- InOrder Traversal
- PostOrder Traversal

Let us consider the below Binary Tree and try to traverse it using the above methods.



GitHub URL: https://github.com/imszainab/DSPracticals

1. Binary Tree PreOrder Traversal

In a PreOrder traversal, the nodes are traversed according to the following sequence from any given node:

- It will mark the current node as visited first.
- Then, if a left child exists, it will go to the left sub-tree and continue the same process.
- After visiting the left sub-tree, it will then move to its right sub-tree and continue the same process.

Since the sequence is node -> left -> right, it is referred to as a PreOrder traversal, since the node is visited before the left sub-tree.

2. Binary Tree InOrder Traversal

In an InOrder traversal, the nodes are traversed according to the following sequence from any given node:

If a left child exists, it will always go to it first.

- After it visits the left sub-tree, it will visit the currently given node
- After visiting the node, it will then move to its right sub-tree.

As the sequence is left -> node -> right, it will refer to it as an InOrder traversal, since we will visit the nodes “in order”, from left to the right.

3. Binary Tree PostOrder Traversal

In a PostOrder traversal, the nodes are traversed according to the following sequence from any given node:

- If a left child exists, it will always go to it first.
- After visiting the left sub-tree, it will then move to its right sub-tree.
- After it visits the right sub-tree, it will finally visit the currently given node

Since the sequence is left -> right -> node, it is referred to as a PostOrder traversal, since the nodes are visited at the last.

CODE –

Practical 8.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 8.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
#8. Write a program for inorder, postorder and preorder traversal of tree.
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):
```

Practical 8.py - E:\Zainab\Sem III\Data Structures\Practicals\Practical 8.py (3.9.0rc2)

File Edit Format Run Options Window Help

```
        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)
```

GitHub URL: https://github.com/imszainab/DSPracticals

```
    root_1.insert(4)
    root_1.insert(13)
    root_1.insert(130)
    root_1.insert(123)
    print("Now Ordering Tree Elements With Insert")
    root_1.PrintTree()
    print("Tree Elements In Pre Order")
    root_1.Printpreorder()
    print("Tree Elements: In Order")
    root_1.Printinorder()
    print("Tree Elements In Post Order")
    root_1.Printpostorder()

```

OUTPUT –