



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: Deepak Kumar Murari Prasad Keshri

Roll No: 308

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 STRUCTURE** for the partial fulfilment of Third Semester of BSc IT during the academic year 2021-22.

The journal work is the original study work that has been duly approved in the year 2021-22 by the undersigned.

External Examiner

Ms. Roshni Singh
(Subject-In-Charge)

Date of Examination:

(College Stamp)

INDEX

Sr.No.	Date	Topic	Page No
1	3-7-2021	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 the Matrix, addition, Multiplication and Transpose operations.	
2	10-7-21	Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked list.	
3	17-7-21	Implement the following for stack. a. Perform stack operation using Array implementation b. Implement Tower of Hanoi. c. WAP to scan a polynomial using linked list and add two polynomial . d. WAP to calculate factorial and to compute the factors of a given no. 1. Using recursion 2. Using iteration.	
4	24-7-21	Perform Queue operation using Circular Array implementation.	
5	31-7-21	write a program to search an element from a list. Give user the option to perform Linear or Binary search.	
6	7-8-21	write a program to sort a list of element. Gives the user the option to perform sorting using Insertion sort, Bubble sort, or selection sort.	

7	14-8-21	<p>Implement the following for Hashing.</p> <ol style="list-style-type: none"> Write a program to implement the collision technique. write a program to implement the concept of linear probing. 	
8	21-8-21	Write program for inorder, postorder and preorder traversal of tree	
9	29-8-21	Write a program for shortest path diagram.	

Repository Link: <https://github.com/deepak101010/-DS-DATA-STRUCTURE>

Practical No:1-A

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:

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

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

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

Basic Operations

Following are the basic operations supported by an array.

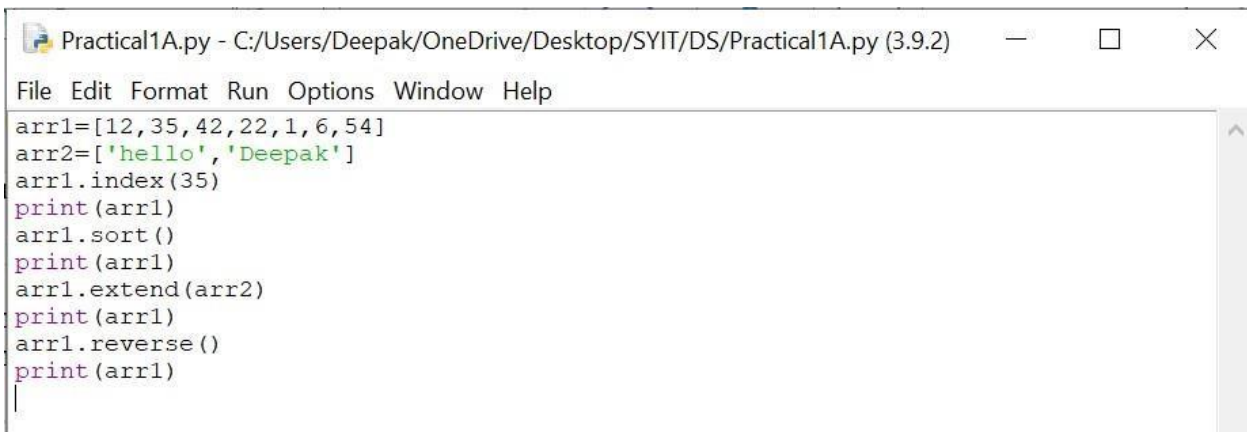
Traverse – print all the array elements one by one.

Insertion – Adds an element at the given index.

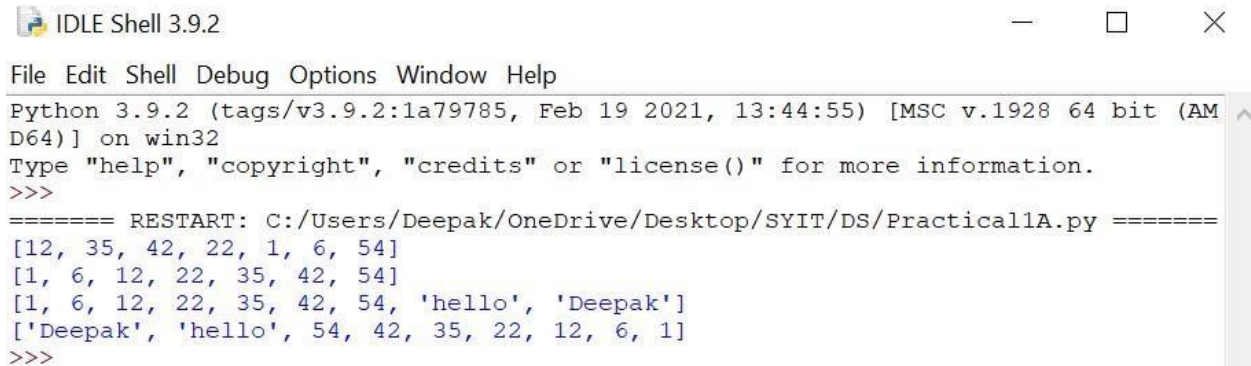
Search – Searches an element using the given index or by the value.

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

Code:



```
Practical1A.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical1A.py (3.9.2)
File Edit Format Run Options Window Help
arr1=[12,35,42,22,1,6,54]
arr2=['hello','Deepak']
arr1.index(35)
print(arr1)
arr1.sort()
print(arr1)
arr1.extend(arr2)
print(arr1)
arr1.reverse()
print(arr1)
```

Output:

```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical1A.py =====
[12, 35, 42, 22, 1, 6, 54]
[1, 6, 12, 22, 35, 42, 54]
[1, 6, 12, 22, 35, 42, 54, 'hello', 'Deepak']
['Deepak', 'hello', 54, 42, 35, 22, 12, 6, 1]
>>>
```

Practical No.: 1-B

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

Theory:

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

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

We can treat each element as a row of the matrix.

For example `X = [[1, 2], [4, 5], [3, 6]]` would represent a 3x2 matrix.

The first row can be selected as `X[0]`. And, the element in first row, first column can be selected as `X[0][0]`.

Multiplication of two matrices `X` and `Y` is defined only if the number of columns in `X` is equal to the number of rows `Y`.

If `X` is $n \times m$ matrix and `Y` is $m \times x$ matrix then, `XY` is defined and has the dimension $n \times x$.
`YX` is not defined (but is defined). Here `Y` is $m \times x$ matrix and `X` is $n \times m$ matrix and is a 1×1 matrix.
 are a couple of ways to implement matrix multiplication in Python.

Code:

```
*Practical1B.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical1B.py (3.9.2)*
File Edit Format Run Options Window Help

X = [[11,7,3],
     [4,5,6],
     [7,8,9]]

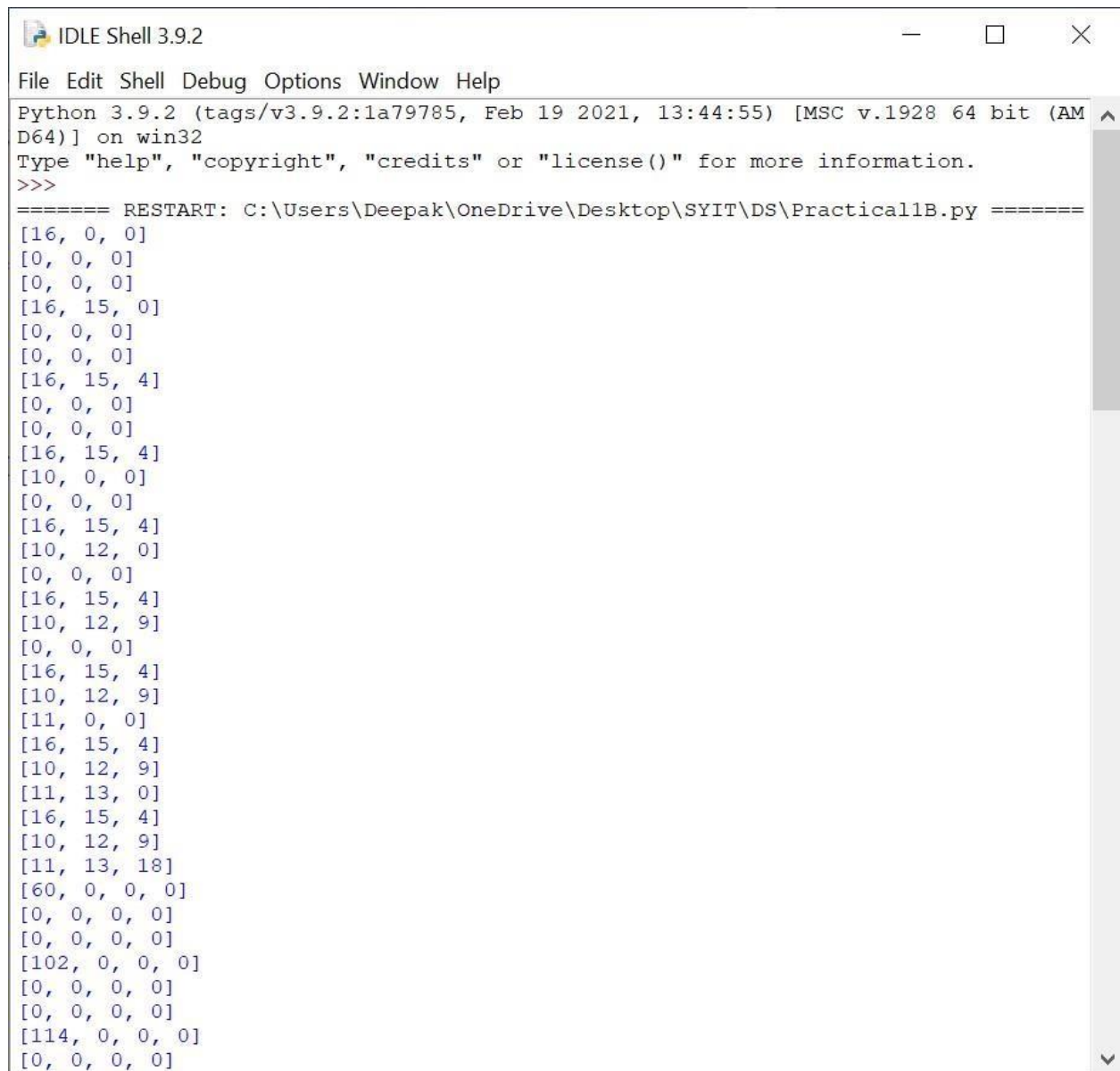
Y = [[5,8,1],
     [6,7,3],
     [4,5,9]]

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

for i in range(len(X)):
    for j in range(len(X[0])):
        result[i][j] = X[i][j] + Y[i][j]
    for r in result:
        print(r)

#3X3 matrix
X = [[12,7,3],
     [4,5,6],
     [7,8,9]]
#4X4 matrix
Y = [[5,8,1,2],
     [6,7,3,0],
     [4,5,9,1]]
result = [[0,0,0,0],
          [0,0,0,0],
          [0,0,0,0]]
#iterate through rows of X
for i in range(len(X)):
    #iterate through columns of Y
    for j in range(len(Y[0])):
        #iterate through rows of Y
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]
        for r in result:
            print(r)
```

Output:



The screenshot shows the IDLE Shell 3.9.2 window. The title bar is 'IDLE Shell 3.9.2'. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The status bar at the bottom indicates 'Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32'. The main text area displays the following output:

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical1B.py =====
[16, 0, 0]
[0, 0, 0]
[0, 0, 0]
[16, 15, 0]
[0, 0, 0]
[0, 0, 0]
[16, 15, 4]
[0, 0, 0]
[0, 0, 0]
[16, 15, 4]
[10, 0, 0]
[0, 0, 0]
[16, 15, 4]
[10, 12, 0]
[0, 0, 0]
[16, 15, 4]
[10, 12, 9]
[0, 0, 0]
[16, 15, 4]
[10, 12, 9]
[11, 0, 0]
[16, 15, 4]
[10, 12, 9]
[11, 13, 0]
[16, 15, 4]
[10, 12, 9]
[11, 13, 18]
[60, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[102, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[114, 0, 0, 0]
[0, 0, 0, 0]
```


Practical No: 2

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

Theory:

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

Code:

```
*Practical2.py - C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical2.py (3.9.2)*
File Edit Format Run Options Window Help
#Name: Deepak SYIT/308

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    def reverse(self):
        prev = None
        current = self.head
        while(current is not None):
            next = current.next
            current.next = prev
            prev = current
            current = next
        self.head = prev

    def insert(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    def printList(self):
        temp = self.head
        while(temp):
            print (temp.data, "-->", end=" ")
            temp = temp.next

l1list = LinkedList()
l1list.insert(20)
l1list.insert(4)
l1list.insert(15)
l1list.insert(85)

print("Given Linked List")
l1list.printList()
l1list.reverse()
print("\nReversed Linked List")
l1list.printList()
```

Output:

```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical2.py =====
Given Linked List
85 --> 15 --> 4 --> 20 -->
Reversed Linked List
20 --> 4 --> 15 --> 85 -->
>>>
```

Practical No: 3-A

Aim: Implement the following for Stack

Theory:

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

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

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

Code:

```
Practiual3A.py - C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practiual3A.py (3...
File Edit Format Run Options Window Help

from sys import maxsize
def createStack():
    stack = []
    return stack
def isEmpty(stack):
    return len(stack) == 0
def push(stack, item):
    stack.append(item)
    print(item + " pushed to stack")

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

def peek(stack):
    if (isEmpty(stack)):
        return str(-maxsize -1)
    return stack[len(stack) -1]
stack = createStack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
print (pop(stack) + "popped from stack")
```

Output:

```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practiual3A.py =====
1 pushed to stack
2 pushed to stack
3 pushed to stack
3popped from stack
>>> |
```

Practical No.: 3-B

Aim: Implement
Tower of Hanoi.

Theory:

Create a tower_of_hanoi recursive and pass two arguments : the number of disks n and the name of the rods such as source, aux, and target.

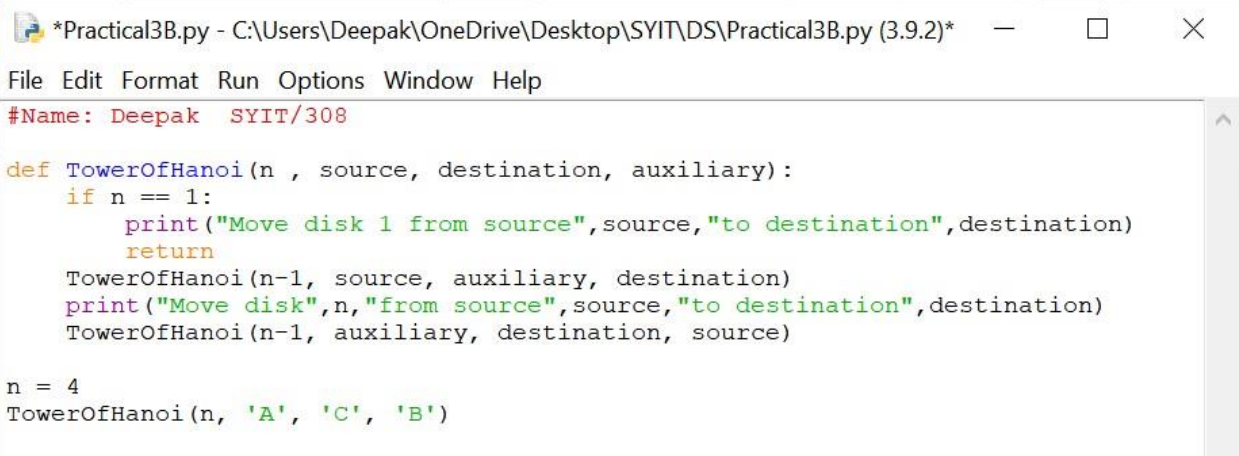
We can define the base case when the number of disks is 1. In this case simply move the one disk from the source to target and return.

Now, move remaining n-1 disks from source to auxiliary using the target as the auxiliary.

Then, the remaining 1 disk move on the source to target.

Move the n-1 disks on the auxiliary to the target using the source as the auxiliary

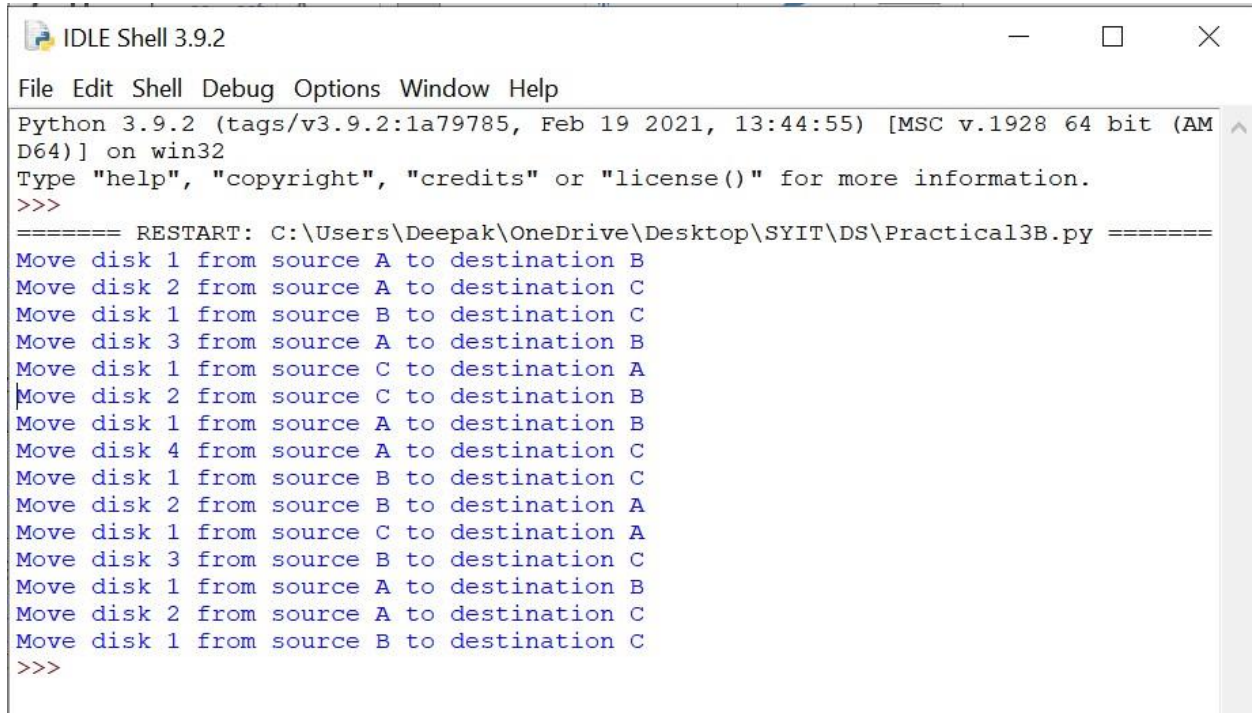
Code:



```
*Practical3B.py - C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical3B.py (3.9.2)*
File Edit Format Run Options Window Help
#Name: Deepak SYIT/308

def TowerOfHanoi(n , source, destination, auxiliary):
    if n == 1:
        print("Move disk 1 from source",source,"to destination",destination)
        return
    TowerOfHanoi(n-1, source, auxiliary, destination)
    print("Move disk",n,"from source",source,"to destination",destination)
    TowerOfHanoi(n-1, auxiliary, destination, source)

n = 4
TowerOfHanoi(n, 'A', 'C', 'B')
```

Output:

```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical3B.py =====
Move disk 1 from source A to destination B
Move disk 2 from source A to destination C
Move disk 1 from source B to destination C
Move disk 3 from source A to destination B
Move disk 1 from source C to destination A
Move disk 2 from source C to destination B
Move disk 1 from source A to destination B
Move disk 4 from source A to destination C
Move disk 1 from source B to destination C
Move disk 2 from source B to destination A
Move disk 1 from source C to destination A
Move disk 3 from source B to destination C
Move disk 1 from source A to destination B
Move disk 2 from source A to destination C
Move disk 1 from source B to destination C
>>>
```

Practical No: 3-C

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

Theory:

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

Code:

```
Practiual3C.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practiual3C.py (3.9...
File Edit Format Run Options Window Help
#Name: Deepak SYIT/308

def add(A, B, m, n):
    size = max(m, n)
    sum = [0 for i in range(size)]
    for i in range(0, m, 1):
        sum[i] = A[i]
    for i in range(n):
        sum[i] += B[i]
    return sum

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

if __name__ == '__main__':

    A = [5, 0, 10, 6]
    B = [1, 2, 4]
    m = len(A)
    n = len(B)

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

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

Output:

```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practiual3C.py =====
First polynomial is
5 + 0X^ 1 + 10X^ 2 + 6X^ 3
Second polynomial is
1 + 2X^ 1 + 4X^ 2
Sum of polynomial is
6 + 2X^ 1 + 14X^ 2 + 6X^ 3
>>> |
```


PRACTICAL NO 3-D (PART 1)

Aim: WAP to calculate factorial and to compute the factors of a given no. (using recursion)

Theory:

Take a number from the user and store it in a variable.

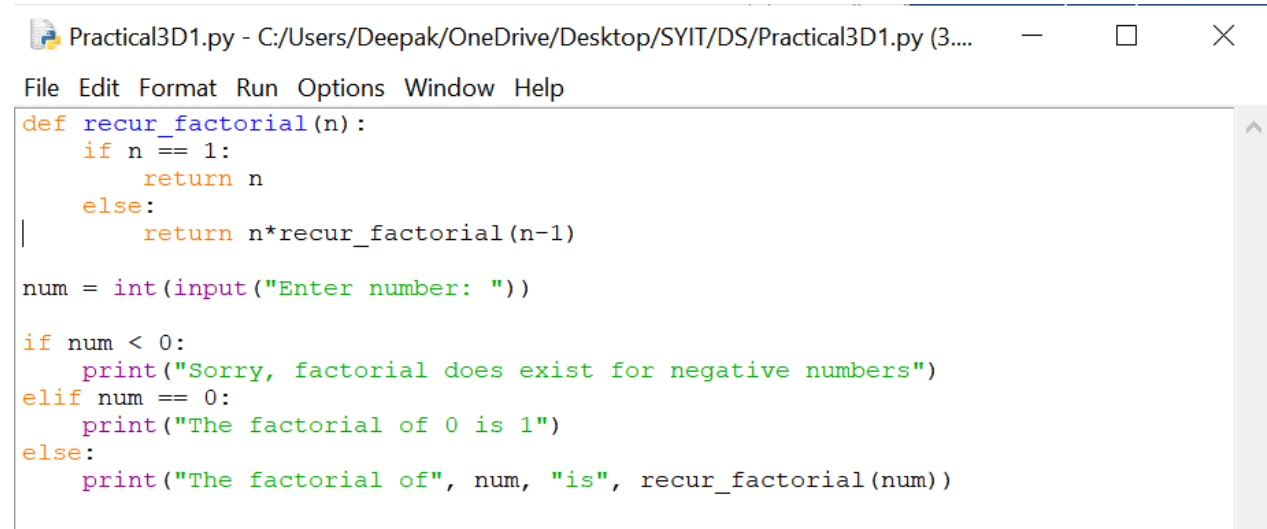
Pass the number as an argument to a recursive factorial function.

Define the base condition as the number to be lesser than or equal to 1 and return 1 if it is.

Otherwise call the function recursively with the number minus 1 multiplied by the number itself

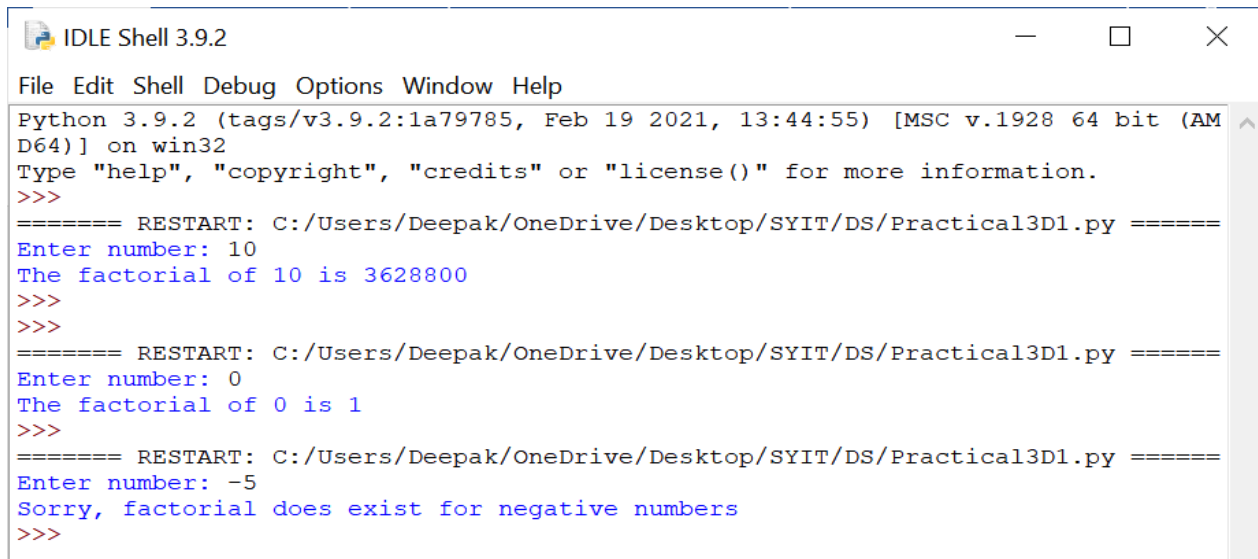
If the number is less than 0 then print the message.

If the number is equal to 0 then factorial is 1 else print factorial of a given number.

Code:

```
Practical3D1.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical3D1.py (3...  
File Edit Format Run Options Window Help  
def recur_factorial(n):  
    if n == 1:  
        return n  
    else:  
        return n*recur_factorial(n-1)  
  
num = int(input("Enter number: "))  
  
if num < 0:  
    print("Sorry, factorial does not exist for negative numbers")  
elif num == 0:  
    print("The factorial of 0 is 1")  
else:  
    print("The factorial of", num, "is", recur_factorial(num))
```

Output:



```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical3D1.py =====
Enter number: 10
The factorial of 10 is 3628800
>>>
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical3D1.py =====
Enter number: 0
The factorial of 0 is 1
>>>
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical3D1.py =====
Enter number: -5
Sorry, factorial does exist for negative numbers
>>>
```

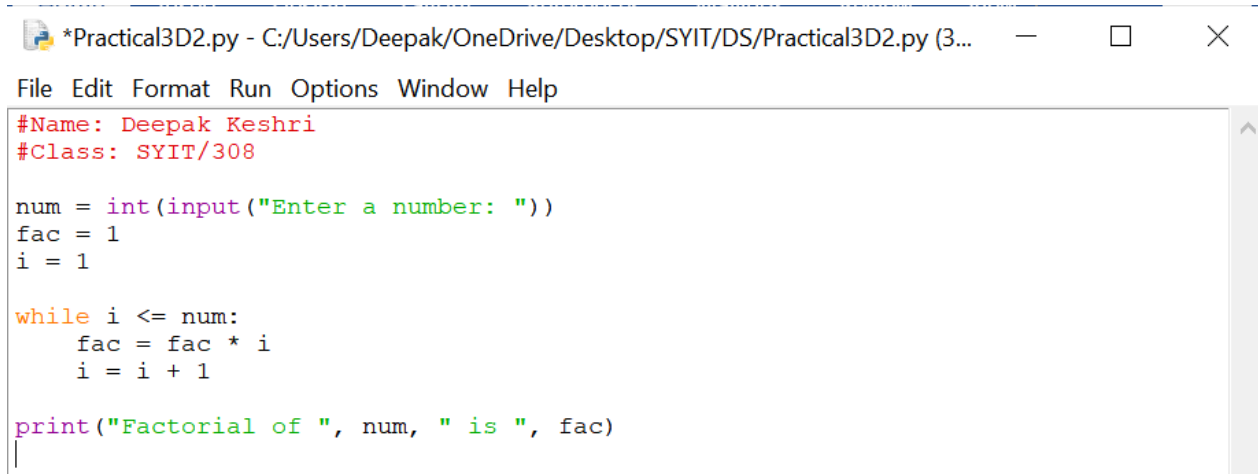
PRACTICAL NO 3-D (PART 2)

Aim: WAP to calculate factorial and to compute the factors of a given no. (using iteration)

Theory:

Iterations are performed through 'for' and 'while' loops. Iterations execute a set of instructions repeatedly until some limiting criteria is met. In contrast to recursion, iteration does not require temporary memory to keep on the results of each iteration.

Code:



The image shows a Python IDLE window titled '*Practical3D2.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical3D2.py (3...'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code is as follows:

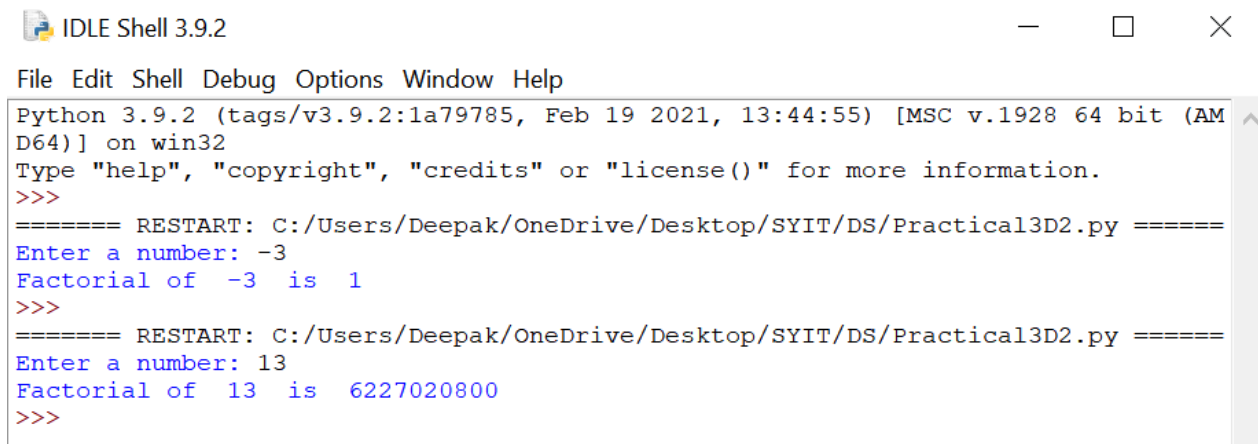
```
#Name: Deepak Keshri
#Class: SYIT/308

num = int(input("Enter a number: "))
fac = 1
i = 1

while i <= num:
    fac = fac * i
    i = i + 1

print("Factorial of ", num, " is ", fac)
```

Output:



The image shows the Python IDLE Shell 3.9.2 window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The output is as follows:

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical3D2.py =====
Enter a number: -3
Factorial of -3 is 1
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical3D2.py =====
Enter a number: 13
Factorial of 13 is 6227020800
>>>
```

Practical No: 4

Aim: Perform Queues operations using Circular Array implementation.

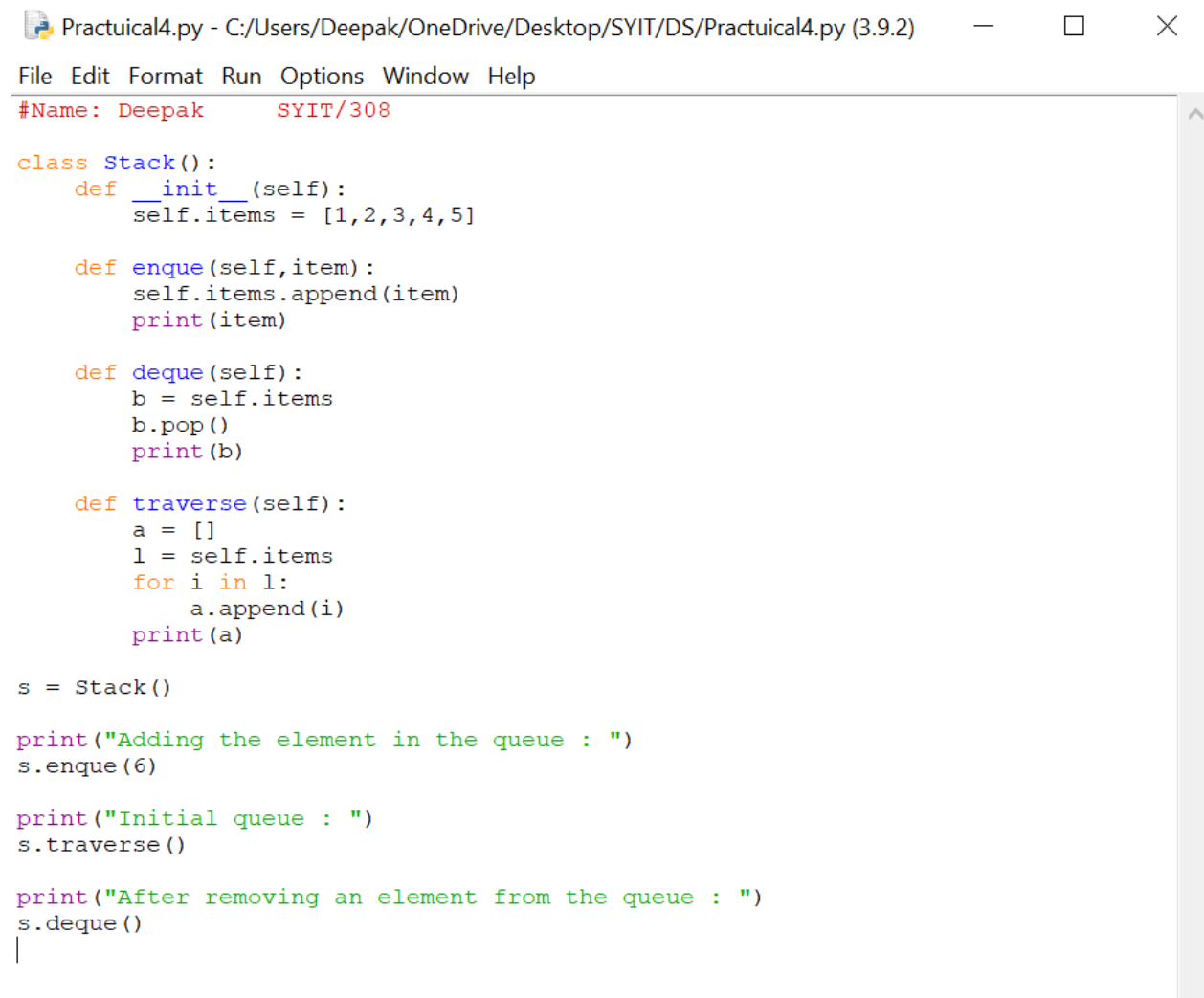
Theory:

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

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

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

Code:



```
Practicaul4.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practicaul4.py (3.9.2)
File Edit Format Run Options Window Help
#Name: Deepak SYIT/308

class Stack():
    def __init__(self):
        self.items = [1,2,3,4,5]

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

    def dequeue(self):
        b = self.items
        b.pop()
        print(b)

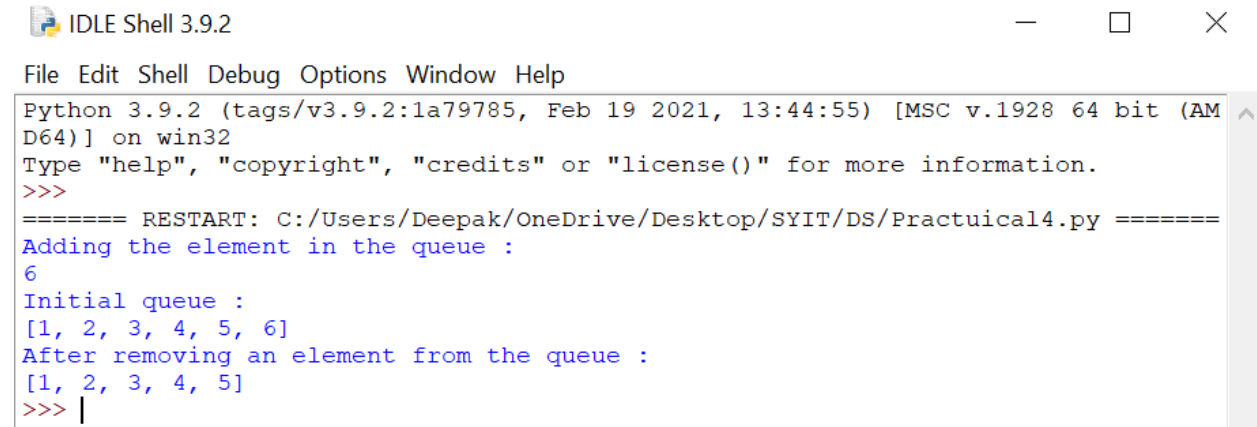
    def traverse(self):
        a = []
        l = self.items
        for i in l:
            a.append(i)
        print(a)

s = Stack()

print("Adding the element in the queue : ")
s.enqueue(6)

print("Initial queue : ")
s.traverse()

print("After removing an element from the queue : ")
s.dequeue()
|
```

Output:

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practuaical4.py =====
Adding the element in the queue :
6
Initial queue :
[1, 2, 3, 4, 5, 6]
After removing an element from the queue :
[1, 2, 3, 4, 5]
>>> |
```

Practical No: 5

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

Theory:

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

Linear Search:

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

Binary Search:

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

Code:

```
*Practical5.py - C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical5.py (3.9.2)*
File Edit Format Run Options Window Help
#Name: Deepak SYIT/308

list1 = [1,2,3,4,5,6,7,8,9,10]
print("List = ",list1)
size = len(list1)
def binary_search(x):
    print("BINARY SEARCHING")
    low = 0
    high = len(list1) - 1
    mid = 0
    while low <= high:
        mid = (high + low) // 2
        if list1[mid] < x:
            low = mid + 1
        elif list1[mid] > x:
            high = mid - 1
        else:
            return mid
    return "None it not in the list"

def linear_search(n):
    print("LINEAR SEARCHING")
    if n not in list1:
        print(n, "not in the list")
    else:
        for i in range(size):
            if list1[i]==n:
                print("index of ", n," is ",i)

n = input("Enter (L) for Linear search and (B) for Binary search :")
if n=="L" or n=="l":
    Y = int(input("Enter a no. from the given list1 "))
    linear_search(Y)
elif n=="B" or n=="b":
    Y = int(input("Enter a no. from the given list1 "))
    print("Index of ",Y," is ",binary_search(Y))
else:
    print("Invalid input")
|
```

Output:

```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical5.py =====
List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Enter (L) for Linear search and (B) for Binary search :b
Enter a no. from the given list1 2
BINARY SEARCHING
Index of 2 is 1
>>>
>>>
===== RESTART: C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical5.py =====
List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Enter (L) for Linear search and (B) for Binary search :l
Enter a no. from the given list1 2
LINEAR SEARCHING
index of 2 is 1
>>> |
```

Practical No: 6

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

Theory:

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

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

Bubble Sort

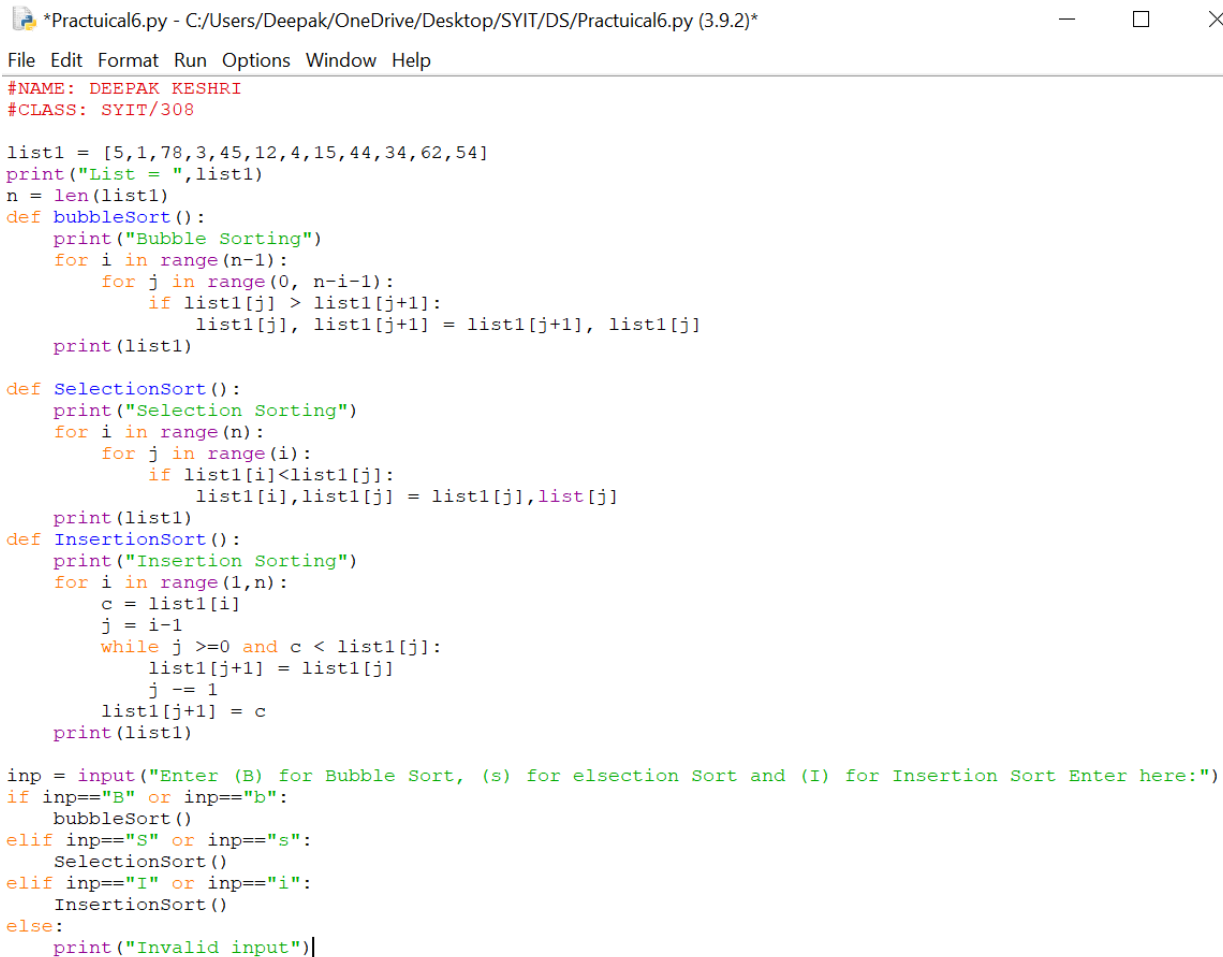
It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

Insertion Sort

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

Selection Sort

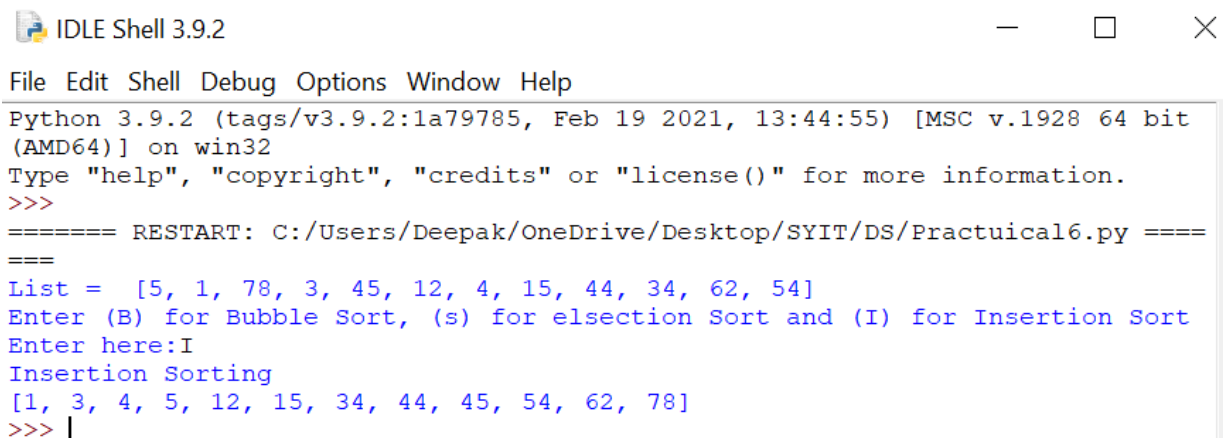
In selection sort we start by finding the minimum value in a given list and move it to a sorted list. Then we repeat the process for each of the remaining elements in the unsorted list. The next element entering the sorted list is compared with the existing elements and placed at its correct position.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

Code:

```
*Practuaical6.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practuaical6.py (3.9.2)*
File Edit Format Run Options Window Help
#NAME: DEEPAK KESHRI
#CLASS: SYIT/308

list1 = [5,1,78,3,45,12,4,15,44,34,62,54]
print("List = ",list1)
n = len(list1)
def bubbleSort():
    print("Bubble Sorting")
    for i in range(n-1):
        for j in range(0, n-i-1):
            if list1[j] > list1[j+1]:
                list1[j], list1[j+1] = list1[j+1], list1[j]
    print(list1)
def SelectionSort():
    print("Selection Sorting")
    for i in range(n):
        for j in range(i):
            if list1[i]<list1[j]:
                list1[i],list1[j] = list1[j],list1[i]
    print(list1)
def InsertionSort():
    print("Insertion Sorting")
    for i in range(1,n):
        c = list1[i]
        j = i-1
        while j >=0 and c < list1[j]:
            list1[j+1] = list1[j]
            j -= 1
        list1[j+1] = c
    print(list1)

inp = input("Enter (B) for Bubble Sort, (s) for elsection Sort and (I) for Insertion Sort Enter here:")
if inp=="B" or inp=="b":
    bubbleSort()
elif inp=="S" or inp=="s":
    SelectionSort()
elif inp=="I" or inp=="i":
    InsertionSort()
else:
    print("Invalid input")
```

Output:

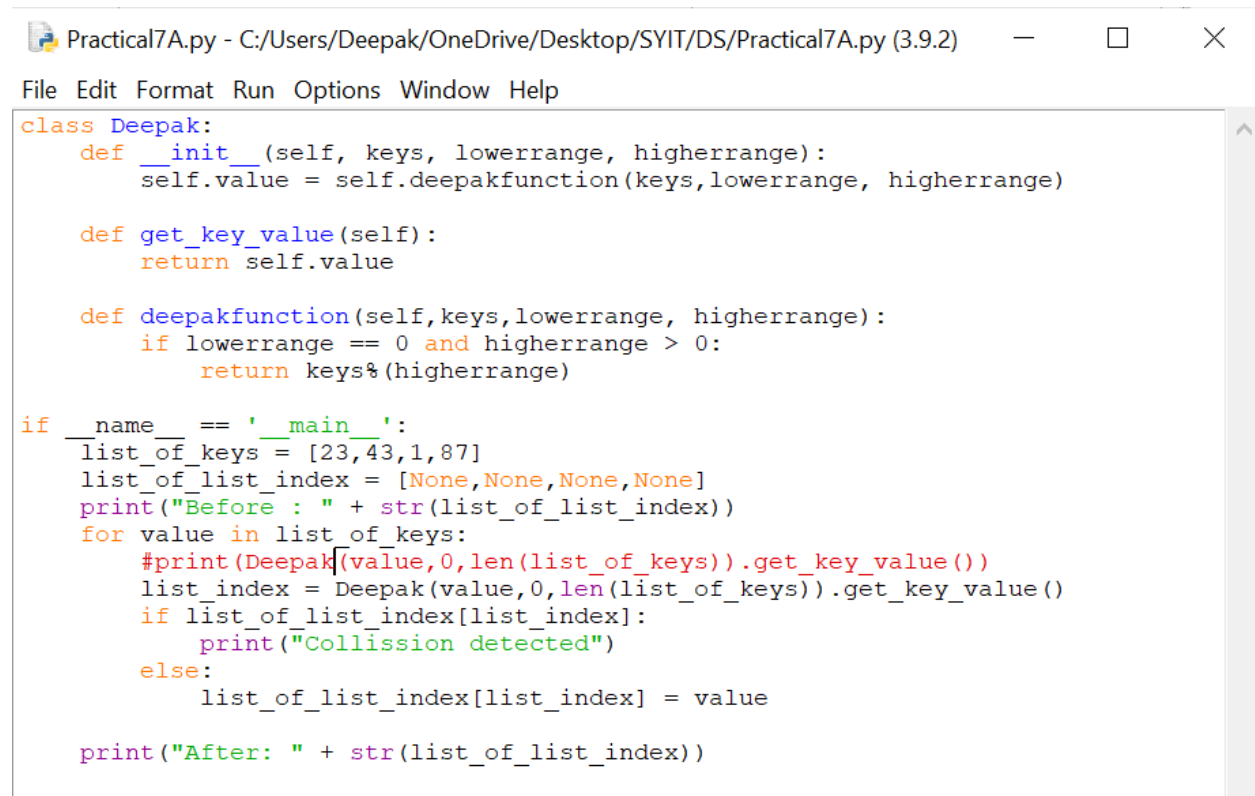
```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practuaical6.py =====
List = [5, 1, 78, 3, 45, 12, 4, 15, 44, 34, 62, 54]
Enter (B) for Bubble Sort, (s) for elsection Sort and (I) for Insertion Sort
Enter here:I
Insertion Sorting
[1, 3, 4, 5, 12, 15, 34, 44, 45, 54, 62, 78]
>>> |
```

Practical No.: 7-A

Aim: Implement the following for Hashing: - write a program to implement the collision technique.

Theory:

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. The most common methods are open addressing, chaining, probabilistic hashing, perfect hashing and coalesced hashing technique.

Code:-

```
Practical7A.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical7A.py (3.9.2)
File Edit Format Run Options Window Help

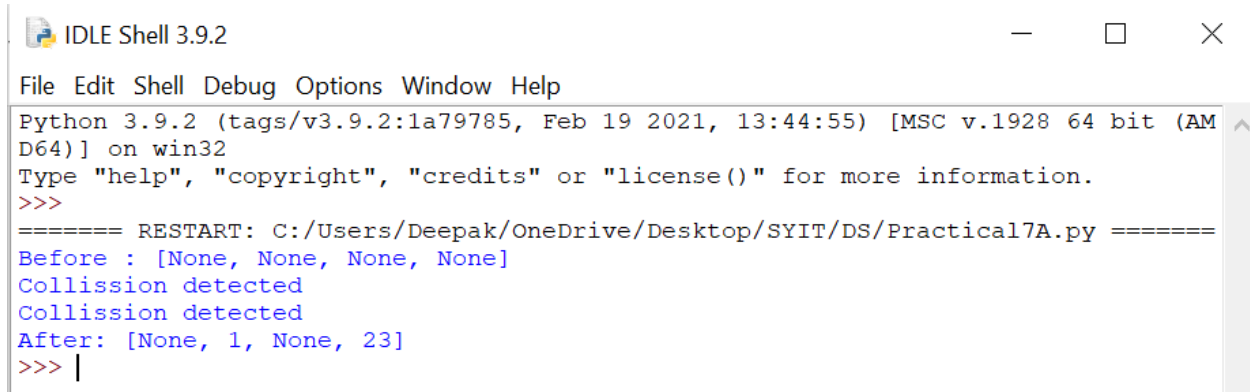
class Deepak:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.deepakfunction(keys, lowerrange, higherrange)

    def get_key_value(self):
        return self.value

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

if __name__ == '__main__':
    list_of_keys = [23, 43, 1, 87]
    list_of_list_index = [None, None, None, None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        #print(Deepak(value, 0, len(list_of_keys)).get_key_value())
        list_index = Deepak(value, 0, len(list_of_keys)).get_key_value()
        if list_of_list_index[list_index]:
            print("Collision detected")
        else:
            list_of_list_index[list_index] = value
    print("After: " + str(list_of_list_index))
```

Output:



```
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical7A.py =====
Before : [None, None, None, None]
Collision detected
Collision detected
After: [None, 23, None, None]
>>> |
```

Practical No.: 7-B

Aim: write a program to implement the concept of linear probing.

Theory:

Linear probing is a component of open addressing schemes for using a hash table to solve the dictionary problem. In the dictionary problem, a data structure should maintain a collection of key–value pairs subject to operations that insert or delete pairs from the collection or that search for the value associated with a given key. In open addressing solutions to this problem, the data structure is an array T (the hash table) whose cells T_i (when nonempty) each store a single key–value pair. A hash function is used to map each key into the cell of T where that key should be stored, typically scrambling the keys so that keys with similar values are not placed near each other in the table. A hash collision occurs when the hash function maps a key into a cell that is already occupied by a different key. Linear probing is a strategy for resolving collisions, by placing the new key into the closest following empty cell.

Code:-

```
*Practical7B.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical7B.py (3.9.2)*
File Edit Format Run Options Window Help

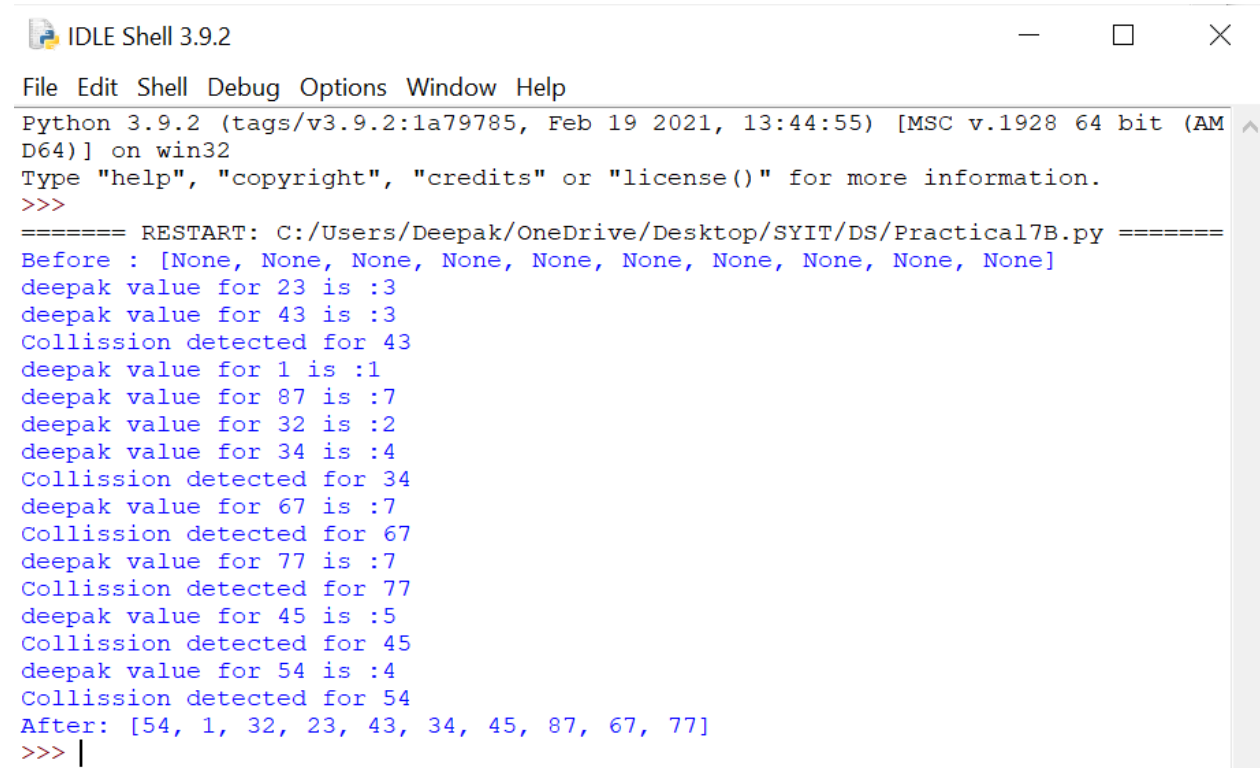
class Deepak:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.deepakfunction(keys, lowerrange, higherrange)

    def get_key_value(self):
        return self.value

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

if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [23, 43, 1, 87, 32, 34, 67, 77, 45, 54]
    list_of_list_index = [None] * len(list_of_keys)
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        # print(deepak(value, 0, len(list_of_keys)).get_key_value())
        list_index = Deepak(value, 0, len(list_of_keys)).get_key_value()
        print("deepak value for " + str(value) + " is : " + str(list_index))
        if list_of_list_index[list_index]:
            print("Collission detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index) - 1:
                    list_index = 0
                else:
                    list_index += 1
                list_full = False
                while list_of_list_index[list_index]:
                    if list_index == old_list_index:
                        list_full = True
                        break
                    if list_index + 1 == len(list_of_list_index):
                        list_index = 0
                    else:
                        list_index += 1
                if list_full:
                    print("List was full . Could not save")
                else:
                    list_of_list_index[list_index] = value
            else:
                list_of_list_index[list_index] = value
    print("After: " + str(list_of_list_index))
```

Output:



```
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical7B.py =====
Before : [None, None, None, None, None, None, None, None, None, None]
deepak value for 23 is :3
deepak value for 43 is :3
Collision detected for 43
deepak value for 1 is :1
deepak value for 87 is :7
deepak value for 32 is :2
deepak value for 34 is :4
Collision detected for 34
deepak value for 67 is :7
Collision detected for 67
deepak value for 77 is :7
Collision detected for 77
deepak value for 45 is :5
Collision detected for 45
deepak value for 54 is :4
Collision detected for 54
After: [54, 1, 32, 23, 43, 34, 45, 87, 67, 77]
>>> |
```

Practical No.: 8

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

Theory:

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

Depth First Traversals:

(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1 Breadth First or Level Order Traversal : 1 2 3 4

5 Please see [this](#) post for Breadth First Traversal. **Inorder Traversal ([Practice](#)):**

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Uses of Inorder

In case of binary search trees (BST), Inorder traversal gives nodes in nondecreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal is reversed can be used.

Example: Inorder traversal for the above-given figure is 4 2 5 1 3. **Preorder Traversal ([Practice](#)):**

Algorithm Preorder(tree)

1. Visit the root.
 2. Traverse the left subtree, i.e., call Preorder(left-subtree)
 3. Traverse the right subtree, i.e., call Preorder(right-subtree)
- Uses of Preorder Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree

Example: Preorder traversal for the above given figure is 1 2 4 5 3. **Postorder Traversal ([Practice](#)):**

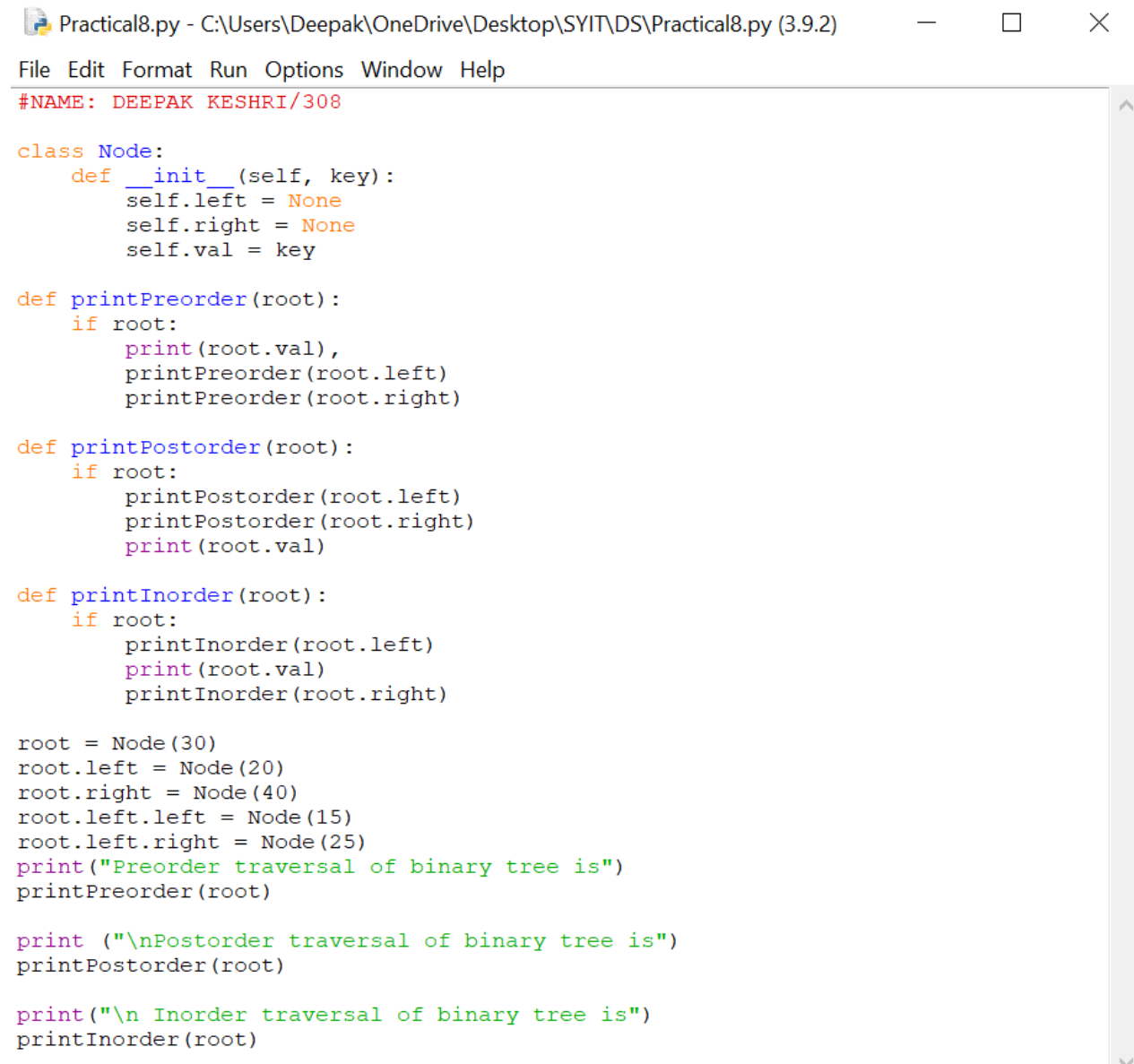
Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)

3. Visit the root.

Uses of Postorder

Postorder traversal is used to delete the tree. Please see [the question for deletion of tree](#) for details. Postorder traversal is also useful to get the postfix expression of an expression tree.

Code:

```
Practical8.py - C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical8.py (3.9.2)
File Edit Format Run Options Window Help
#NAME: DEEPAK KESHRI/308

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

def printPreorder(root):
    if root:
        print(root.val),
        printPreorder(root.left)
        printPreorder(root.right)

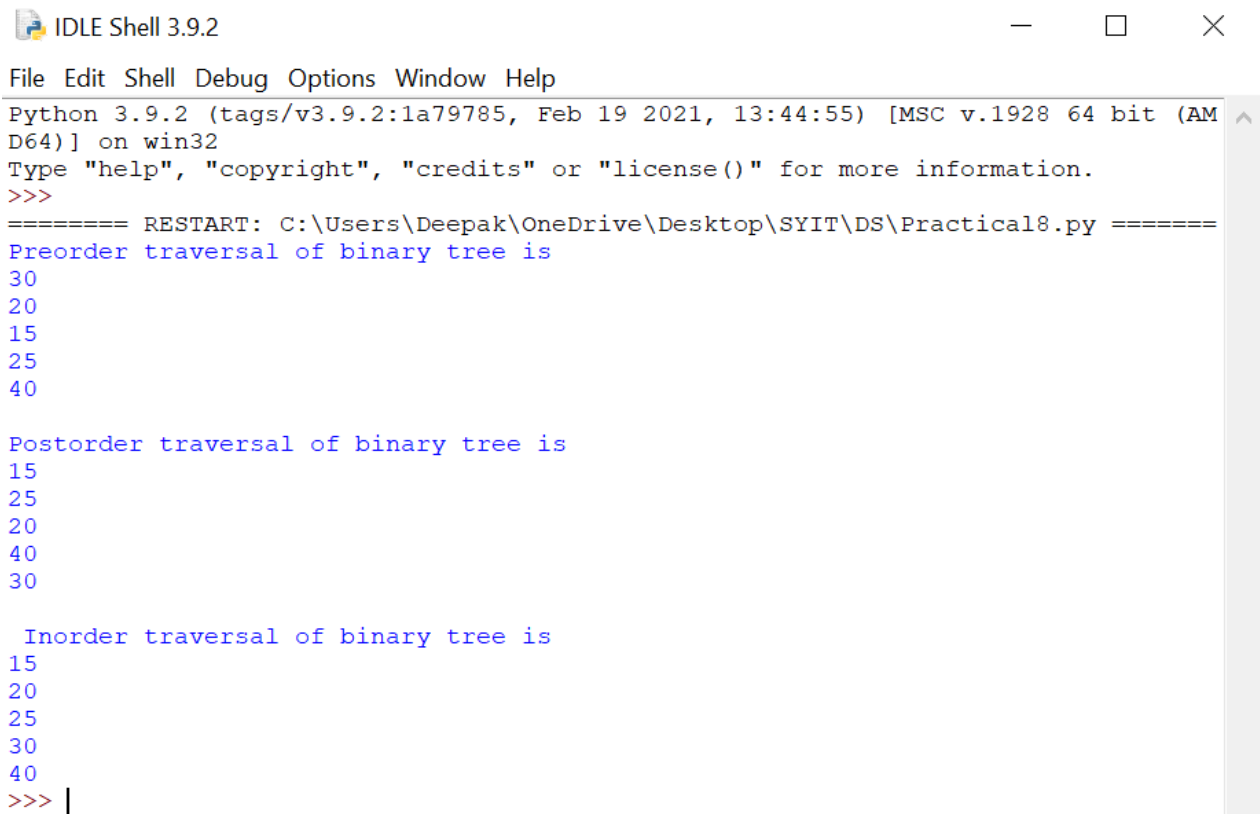
def printPostorder(root):
    if root:
        printPostorder(root.left)
        printPostorder(root.right)
        print(root.val)

def printInorder(root):
    if root:
        printInorder(root.left)
        print(root.val)
        printInorder(root.right)

root = Node(30)
root.left = Node(20)
root.right = Node(40)
root.left.left = Node(15)
root.left.right = Node(25)
print("Preorder traversal of binary tree is")
printPreorder(root)

print("\nPostorder traversal of binary tree is")
printPostorder(root)

print("\n Inorder traversal of binary tree is")
printInorder(root)
```

Output:

```

IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Deepak\OneDrive\Desktop\SYIT\DS\Practical8.py =====
Preorder traversal of binary tree is
30
20
15
25
40

Postorder traversal of binary tree is
15
25
20
40
30

Inorder traversal of binary tree is
15
20
25
30
40
>>> |
```


Practical No.: 9

Aim: Write a program for shortest path diagram.

Theory:

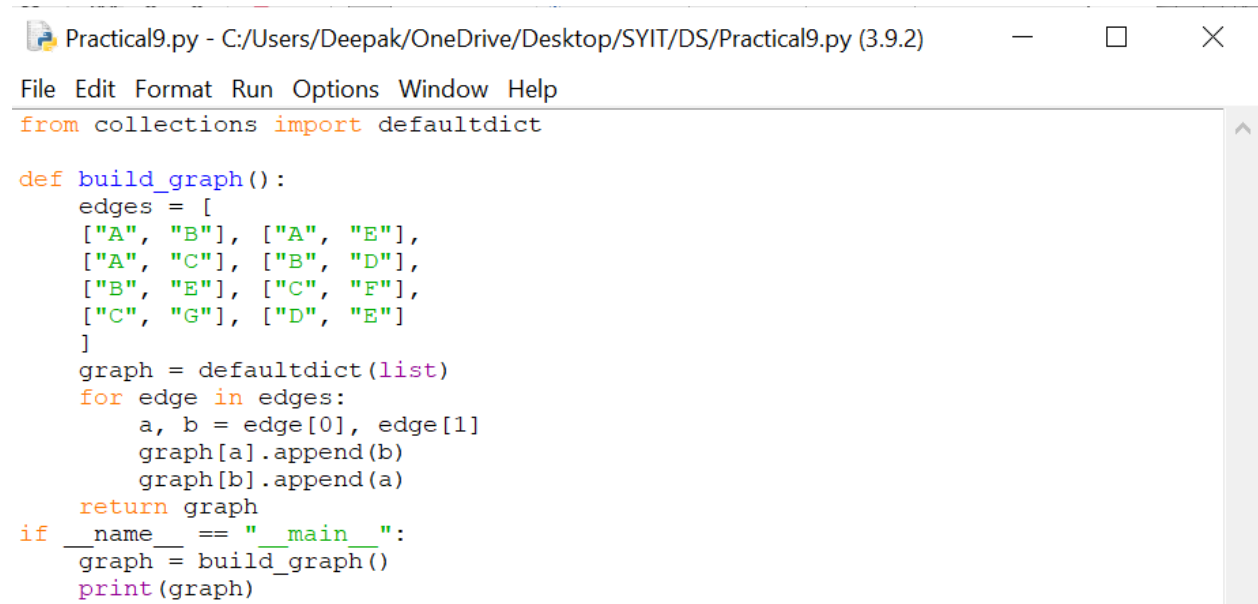
1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3) While *sptSet* doesn't include all vertices:

- Pick a vertex *u* which is not there in *sptSet* and has minimum distance value.
- Include *u* to *sptSet*.

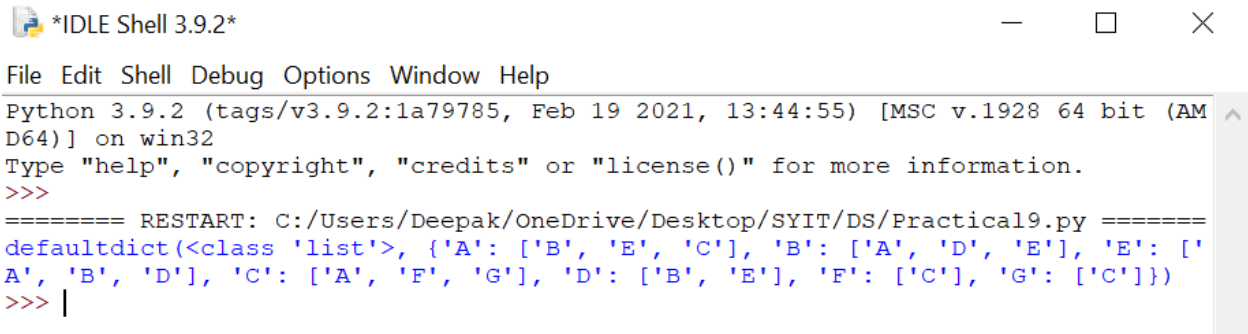
Update distance value of all adjacent vertices of *u*. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex *v*, if the sum of a distance value of *u* (from source) and weight of edge *u-v*, is less than the distance value of *v*, then update the distance value of *v*

Code:

```
Practical9.py - C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical9.py (3.9.2)
File Edit Format Run Options Window Help
from collections import defaultdict

def build_graph():
    edges = [
        ["A", "B"], ["A", "E"],
        ["A", "C"], ["B", "D"],
        ["B", "E"], ["C", "F"],
        ["C", "G"], ["D", "E"]
    ]
    graph = defaultdict(list)
    for edge in edges:
        a, b = edge[0], edge[1]
        graph[a].append(b)
        graph[b].append(a)
    return graph

if __name__ == "__main__":
    graph = build_graph()
    print(graph)
```

Output:

```
*IDLE Shell 3.9.2*
File Edit Shell Debug Options Window Help
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Deepak/OneDrive/Desktop/SYIT/DS/Practical9.py =====
defaultdict(<class 'list'>, {'A': ['B', 'E', 'C'], 'B': ['A', 'D', 'E'], 'E': ['A', 'B', 'D'], 'C': ['A', 'F', 'G'], 'D': ['B', 'E'], 'F': ['C'], 'G': ['C']})
>>> |
```