



MALAD KANDIVALI EDUCATION SOCIETY'S

**NAGINDAS KHANDWALA COLLEGE OF COMMERCE, ARTS &
MANAGEMENT STUDIES & SHANTABEN NAGINDASKHANDWALA
COLLEGE OF SCIENCE
MALAD [W], MUMBAI – 64
AUTONOMOUS INSTITUTION
(Affiliated To University Of Mumbai)
Reaccredited 'A' Grade by NAAC | ISO 9001:2015 Certified**

CERTIFICATE

Name: Mr. _____ KANOJIA KAMLESH ARVIND _____

Roll No: 324 Programme: BSc CS 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)

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) 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:-1A

AIM:- 1a) 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.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical1a.py>

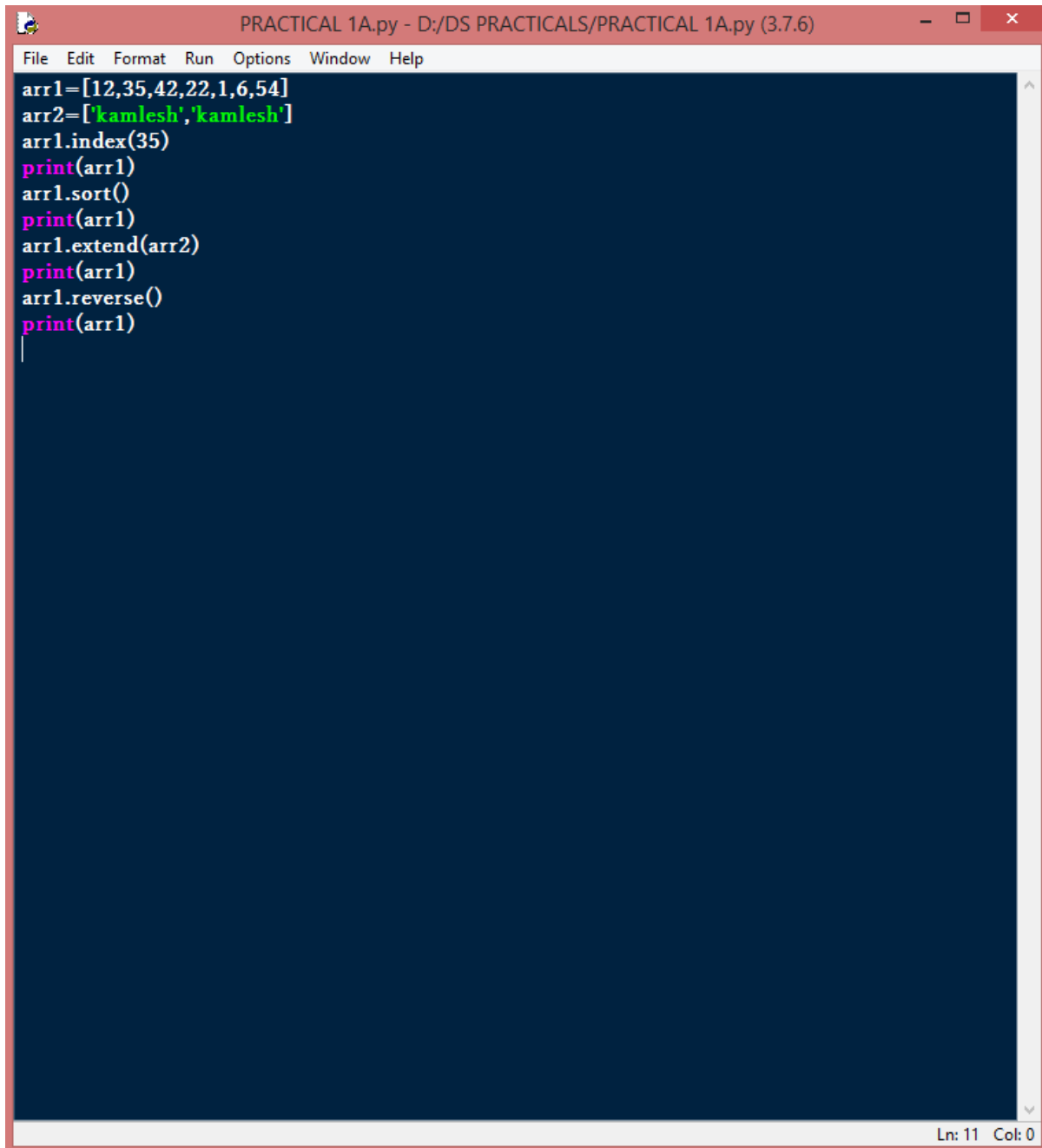
THEORY: Searching: Searching is a very basic necessity when you store data in different data structures. The simplest approach is to go across every element in the data structure and match it with the value you are searching for. This is known as linear search. It is inefficient and rarely used, but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

Sorting: 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.

Merging: Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being $O(n \log n)$, it is one of the most respected algorithms. Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Reversing: reverse () is an inbuilt method in Python programming language that reverses objects of list in place. Returns: The reverse () method does not return any value but reverse the given object from the list

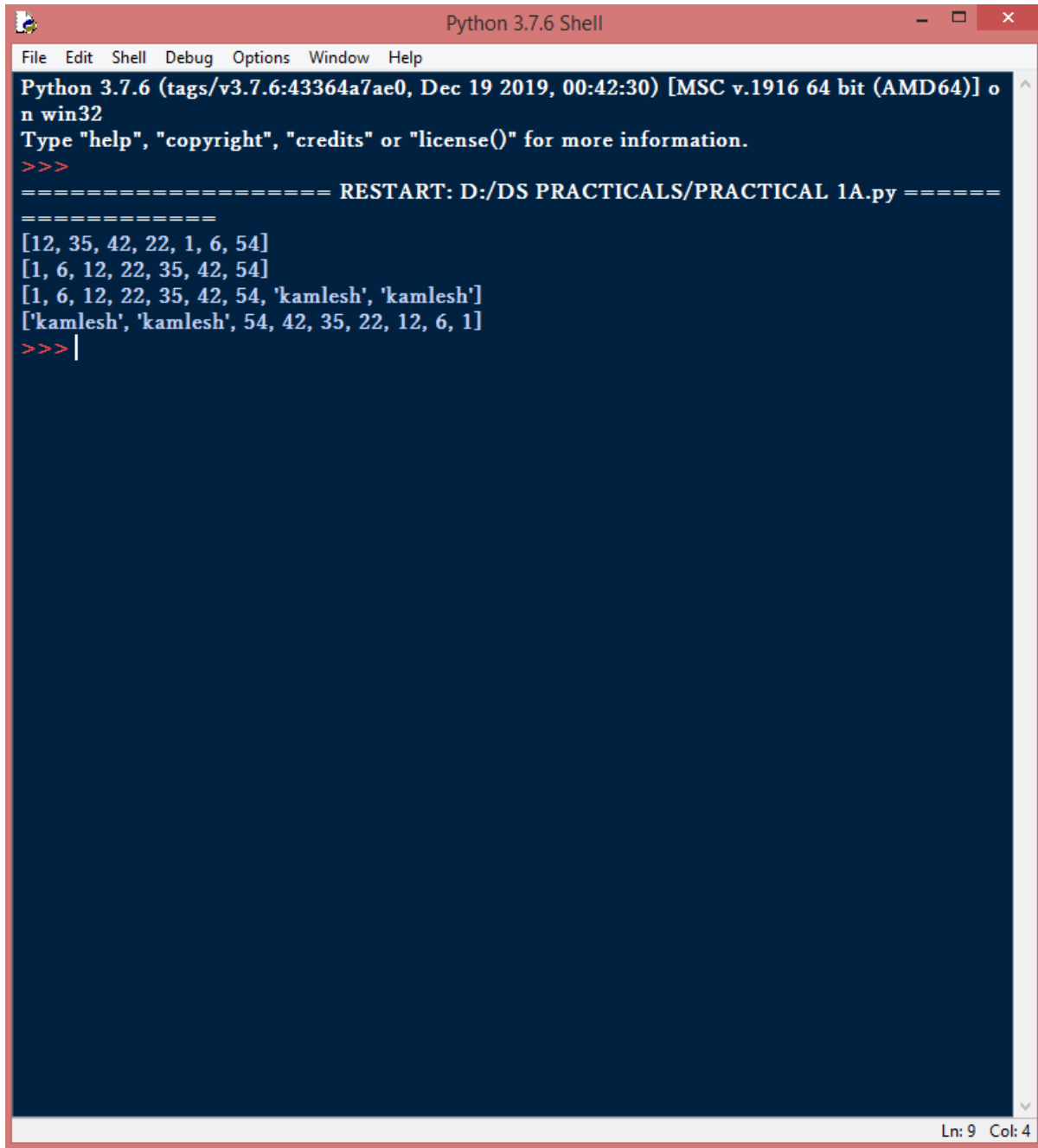
CODE:



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

Ln: 11 Col: 0

OUTPUT:



```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 1A.py =====
=====
[12, 35, 42, 22, 1, 6, 54]
[1, 6, 12, 22, 35, 42, 54]
[1, 6, 12, 22, 35, 42, 54, 'kamlesh', 'kamlesh']
['kamlesh', 'kamlesh', 54, 42, 35, 22, 12, 6, 1]
>>> |
```

Ln: 9 Col: 4

PRACTICAL NO:-1B

AIM:-1b) Write the program to perform the matrix addition, Multiplication and Transpose operation.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%201b.py>

Theory:

Matrix Addition: In Python, we can implement a matrix as a 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. First row can be selected as $X[0]$ and the element in first row, first column can be selected as $X[0][0]$.

We can perform matrix addition in various ways in Python:

1. Matrix Addition using Nested Loop.
2. Matrix Addition using Nested List Comprehension.

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

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

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

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

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

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

We can perform matrix multiplication in various ways in Python:

1. Matrix Multiplication using Nested Loop.
2. Matrix Multiplication using Nested List Comprehension.

Matrix Transpose: In Python, we can implement a matrix as a 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 the first-row first column can be selected as $X[0][0]$.

Transpose of a matrix is the interchanging of rows and columns. It is denoted as X' . The element at ith row and jth column in X will be placed at jth row and ith column in X' . So if X is a 3x2 matrix, X' will be a 2x3 matrix.

We can perform matrix transpose various ways in Python:

1. Matrix Transpose using Nested Loop.
2. Matrix Transpose using Nested List Comprehension.

CODE:

```
PRACTICAL 1B.py - D:\DS PRACTICALS\PRACTICAL 1B.py (3.7.6)
File Edit Format Run Options Window Help
# Program to add two matrices
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]]

# iterate through rows
for i in range(len(X)):
    # iterate through columns
    for j in range(len(X[0])):
        result[i][j] = X[i][j] + Y[i][j]
print("ADDITION_OF_TWO_MATRIX")
for r in result:
    print(r)

# Program to multiply two matrices
# 3x3 matrix
X = [[12,7,3],
     [4 ,5,6],
     [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
     [6,7,3,0],
     [4,5,9,1]]
# result is 3x4
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])):
```

Ln: 1 Col: 0

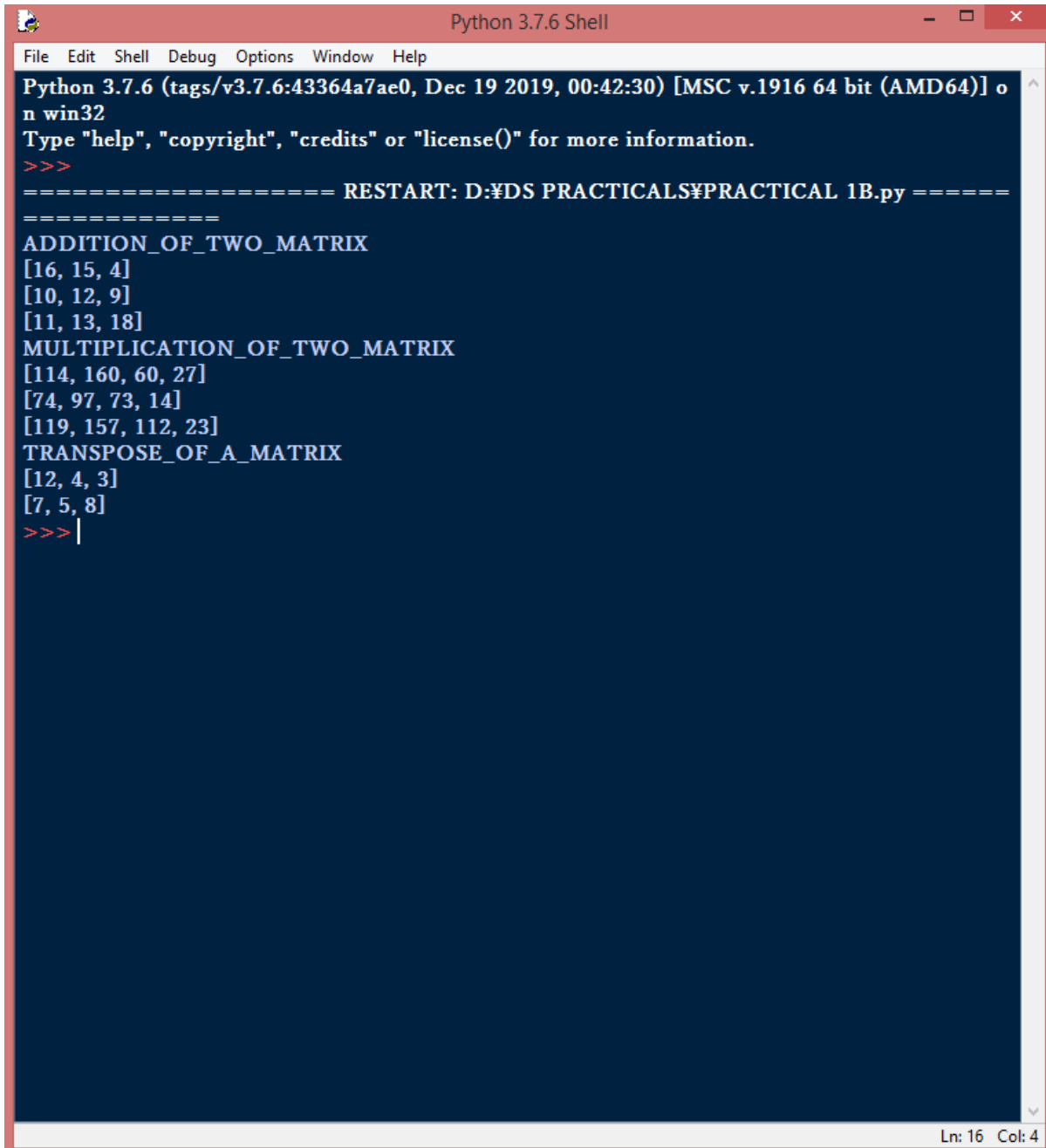
```
PRACTICAL 1B.py - D:\DS PRACTICALS\PRACTICAL 1B.py (3.7.6)
File Edit Format Run Options Window Help

# Program to multiply two matrices
# 3x3 matrix
X = [[12,7,3],
     [4 ,5,6],
     [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
     [6,7,3,0],
     [4,5,9,1]]
# result is 3x4
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]
print("MULTIPLICATION_OF_TWO_MATRIX")
for r in result:
    print(r)

# Program to transpose a matrix
X = [[12,7],
     [4 ,5],
     [3 ,8]]
result = [[0,0,0],
          [0,0,0]]
# iterate through rows
for i in range(len(X)):
# iterate through columns
    for j in range(len(X[0])):
        result[j][i] = X[i][j]
print("TRANSPOSE_OF_A_MATRIX")
for r in result:
    print(r)
```

Ln: 1 Col: 0

OUTPUT:



```
Python 3.7.6 (tags/v3.7.6:43364a7ac0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\PDS PRACTICALS\PRACTICAL 1B.py =====
=====
ADDITION_OF_TWO_MATRIX
[16, 15, 4]
[10, 12, 9]
[11, 13, 18]
MULTIPLICATION_OF_TWO_MATRIX
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
TRANSPOSE_OF_A_MATRIX
[12, 4, 3]
[7, 5, 8]
>>> |
```

Ln: 16 Col: 4

PRACTICAL NO:-2

AIM:-2 Implement Linked List. Include options for Insertion, deletion and search of a number, reverse the list and concatenate the lists.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical2.py>

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. 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.

Searching: 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. **Sorting:** 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.

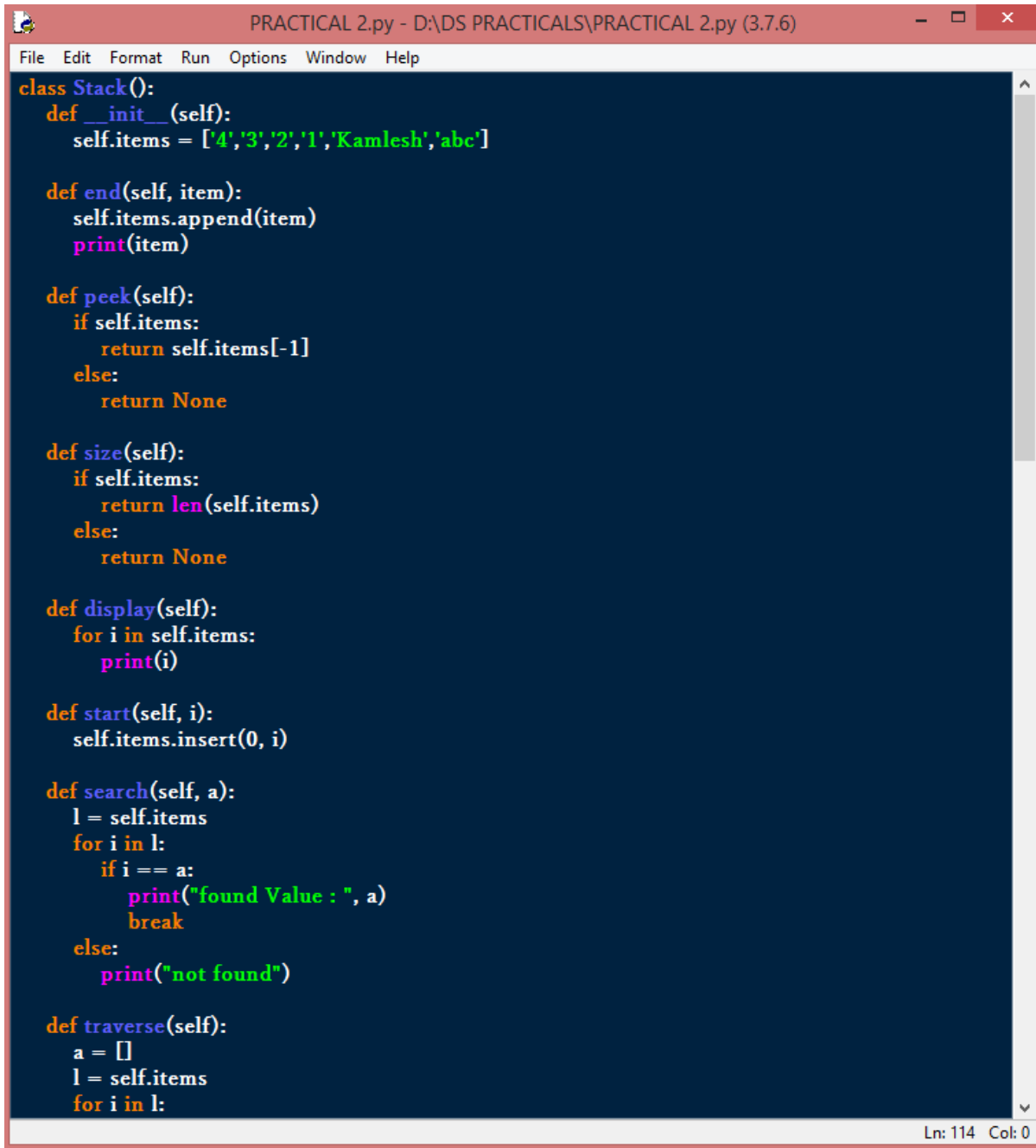
Merging: Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being $O(n \log n)$, it is one of the most respected algorithms. Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Reversing: `reverse ()` is an inbuilt method in Python programming language that reverses objects of list in place. **Returns:** The `reverse ()` method does not return any value but reverse the given object from the list

Iteration: In Python, the iterative statements are also known as looping statements or repetitive statements. The iterative statements are used to execute a part of the program repeatedly as long as a given condition is true.

Recursion: Recursion can also be seen as self-referential function composition. We apply a function to an argument, then pass that result on as an argument to a second application of the same function, and so on.

CODE:



```
class Stack():
    def __init__(self):
        self.items = ['4','3','2','1','Kamlesh','abc']

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

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

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

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

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

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

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

Ln: 114 Col: 0

```
PRACTICAL 2.py - D:\DS PRACTICALS\PRACTICAL 2.py (3.7.6)
File Edit Format Run Options Window Help

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

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

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

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

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

Ln: 114 Col: 0
```

```
*PRACTICAL 2.py - D:\DS PRACTICALS\PRACTICAL 2.py (3.7.6)*
File Edit Format Run Options Window Help

self.items1 = ['4','3','2','1','6']

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

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

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

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

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

Ln: 109 Col: 0

OUTPUT:

```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on
n win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\PRACTICALS\PRACTICAL 2.py =====
=====
-1
6
7
search the specific value :
found Value : -2
Display the values one by one :
-2
-1
5
-2
4
3
2
1
Kamlesh
abc
-1
6
7
peek (End Value) : 7
treverse the values :
['-2', '-1', '5', '-2', '4', '3', '2', '1', 'Kamlesh', 'abc', '-1', '6', '7']
Shotting the values :
['-1', '-1', '-2', '-2', '1', '2', '3', '4', '5', '6', '7', 'Kamlesh', 'abc']
Reversing the values :
['abc', 'Kamlesh', '7', '6', '5', '4', '3', '2', '1', '-2', '-2', '-1', '-1']
remove value from particular index which is defined earlier
['-1', '-2', '-2', '1', '2', '3', '4', '5', '6', '7', 'Kamlesh', 'abc']
merge
['1', '1', '2', '2', '3', '3', '4', '4', '6', 'Kamlesh', 'abc']
>>> |
```

Ln: 35 Col: 4

PRACTICAL NO:-3A

AIM:-3a) Perform Stack Operations using Array Implementation

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%203a.py>

Theory:

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

push(value) - Inserting value into the stack

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at top position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

- Step 1 - Check whether stack is FULL. (top == SIZE-1)
- Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

pop() - Delete a value from the Stack

In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from top position. Pop function does not take any value as parameter. We can use the following steps to pop an element from the stack...

- Step 1 - Check whether stack is EMPTY. (top == -1)
- Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).

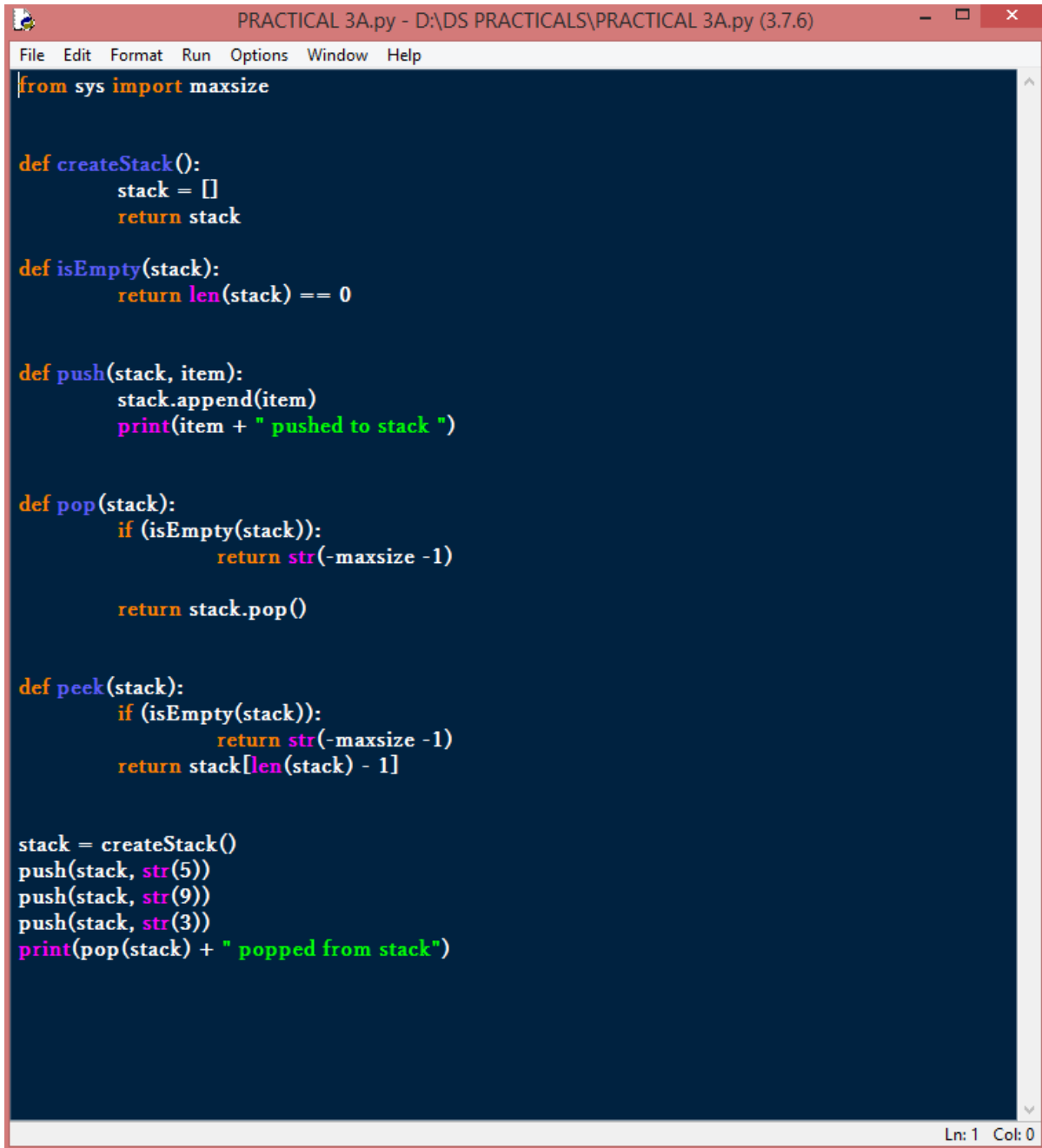
display() - Displays the elements of a Stack

We can use the following steps to display the elements of a stack...

- Step 1 - Check whether stack is EMPTY. (top == -1)
- Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.

- Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
- Step 3 - Repeat above step until i value becomes '0'.

CODE:



```
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)

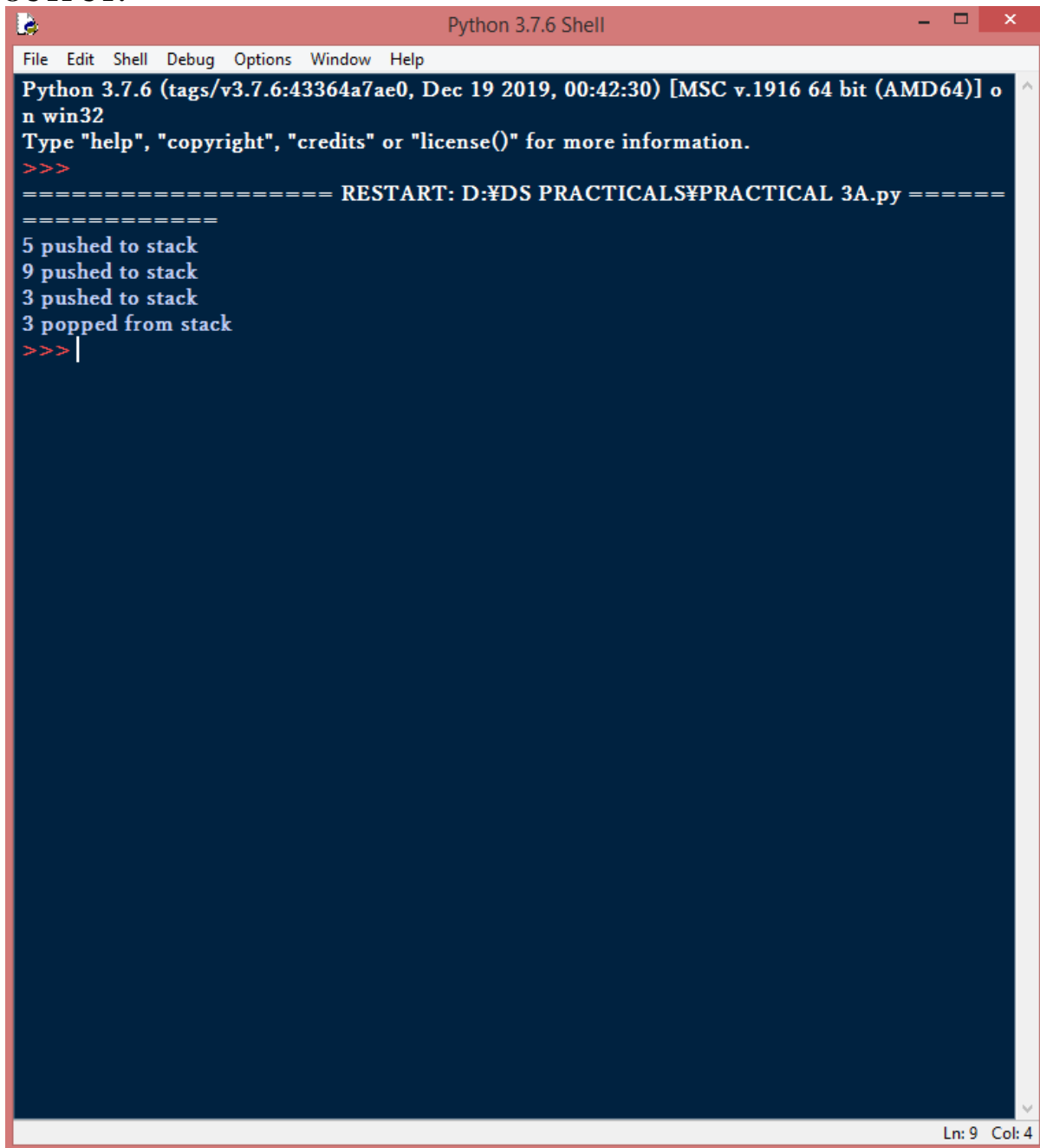
    return stack.pop()

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

stack = createStack()
push(stack, str(5))
push(stack, str(9))
push(stack, str(3))
print(pop(stack) + " popped from stack")
```

Ln: 1 Col: 0

OUTPUT:



The screenshot shows a Python 3.7.6 Shell window with a dark blue background and white text. The window title is "Python 3.7.6 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The output text is as follows:

```
Python 3.7.6 (tags/v3.7.6:43364a7ac0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\PDS PRACTICALS\PRACTICAL 3A.py =====
=====
5 pushed to stack
9 pushed to stack
3 pushed to stack
3 popped from stack
>>> |
```

The status bar at the bottom right indicates "Ln: 9 Col: 4".

PRACTICAL NO:-3C

AIM:-3c) WAP to scan a polynomial using linked list and add two polynomial.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%203c.py>

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:

```
PRACTICAL 3C.py - D:\DS PRACTICALS\PRACTICAL 3C.py (3.7.6)
File Edit Format Run Options Window Help
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 = [7, 2, 0, 15]

    B = [4, 8, 3]
    m = len(A)
    n = len(B)

    print("First polynomial is")
    printPoly(A, m)
    print("\n", end = "")
    print("Second polynomial is")
```

Ln: 48 Col: 0

```
PRACTICAL 3C.py - D:\DS PRACTICALS\PRACTICAL 3C.py (3.7.6)
File Edit Format Run Options Window Help

    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 = [7, 2, 0, 15]

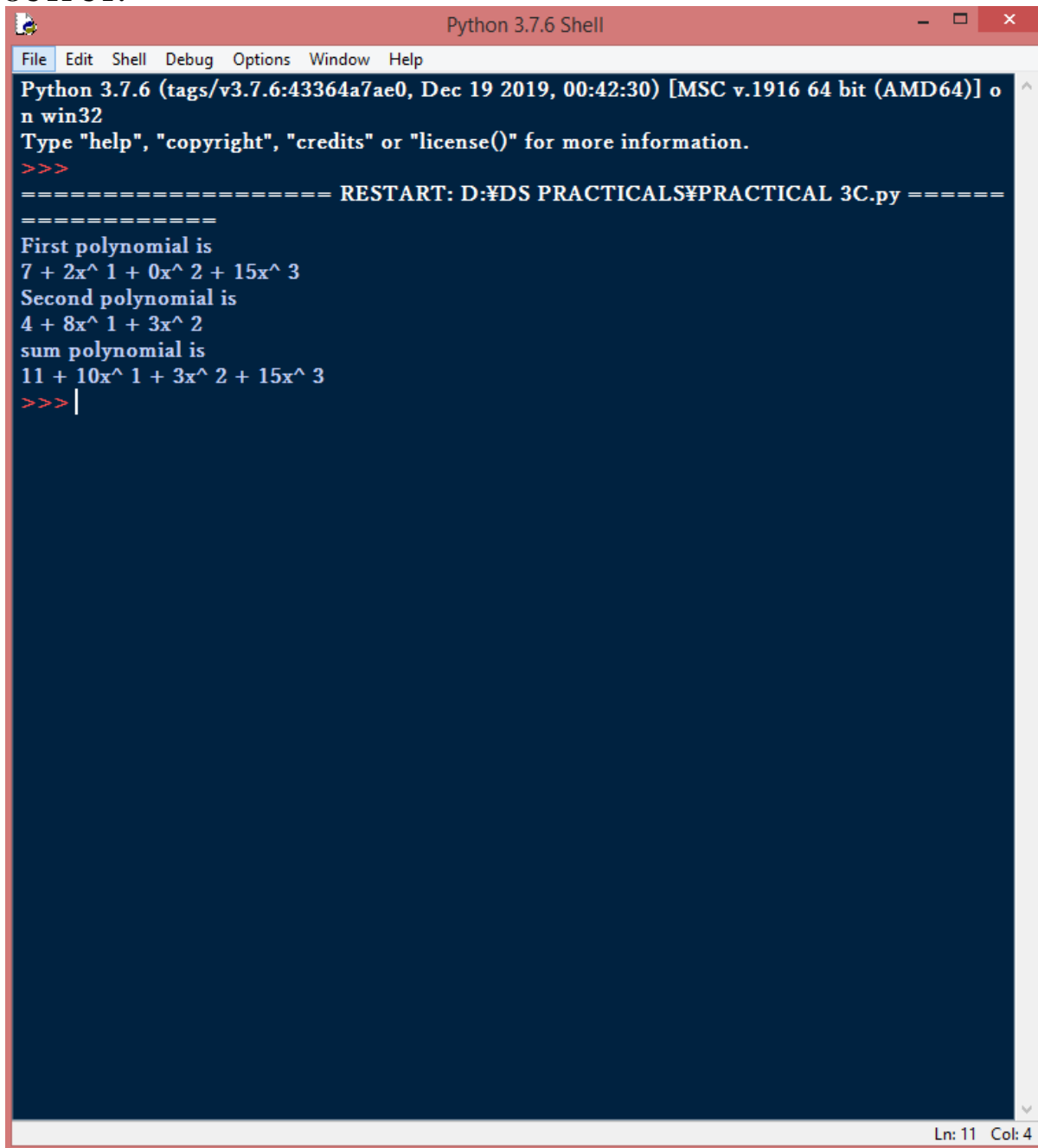
    B = [4, 8, 3]
    m = len(A)
    n = len(B)

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

Ln: 48 Col: 0

OUTPUT:



```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\PRACTICALS\PRACTICAL 3C.py =====
First polynomial is
7 + 2x^1 + 0x^2 + 15x^3
Second polynomial is
4 + 8x^1 + 3x^2
sum polynomial is
11 + 10x^1 + 3x^2 + 15x^3
>>> |
```

Ln: 11 Col: 4

PRACTICAL NO:-3D

AIM:-3d) WAP to Calculate Factorial and to Compute the Factors of the given no
i) Using Recursion. ii) Using Iteration.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%203d.py>

Theory:

The factorial of a number is the product of all the integers from 1 to that number.

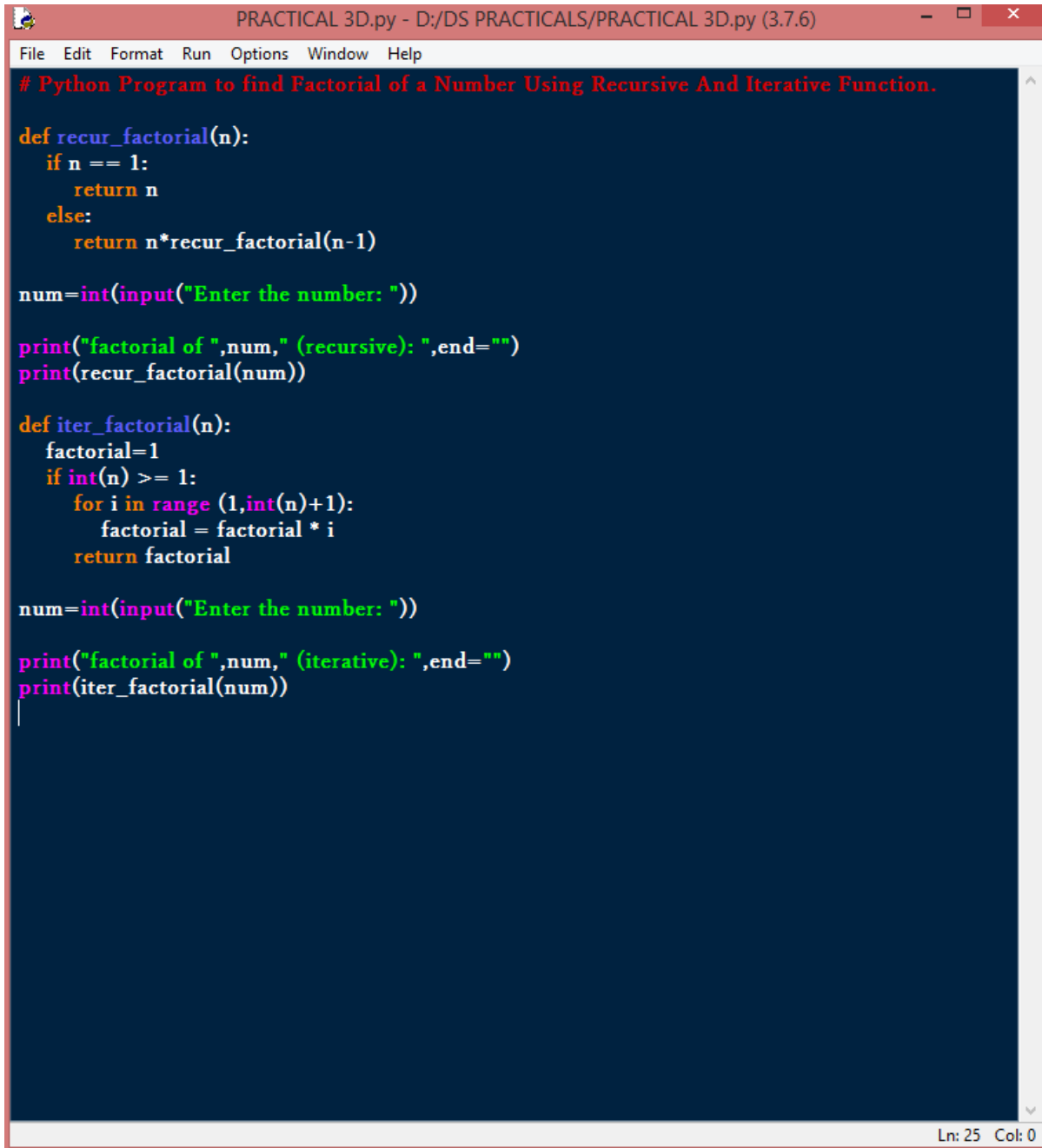
For example, the factorial of 6 is $1*2*3*4*5*6 = 720$. Factorial is not defined for negative numbers and the factorial of zero is one, $0! = 1$.

Iterative Solution:

Factorial can also be calculated iteratively as recursion can be costly for large numbers. Here we have shown the iterative approach using both for and while loop.

Using For loop

CODE:



```
PRACTICAL 3D.py - D:/DS PRACTICALS/PRACTICAL 3D.py (3.7.6)
File Edit Format Run Options Window Help
# Python Program to find Factorial of a Number Using Recursive And Iterative Function.

def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)

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

print("factorial of ",num," (recursive): ",end="")
print(recur_factorial(num))

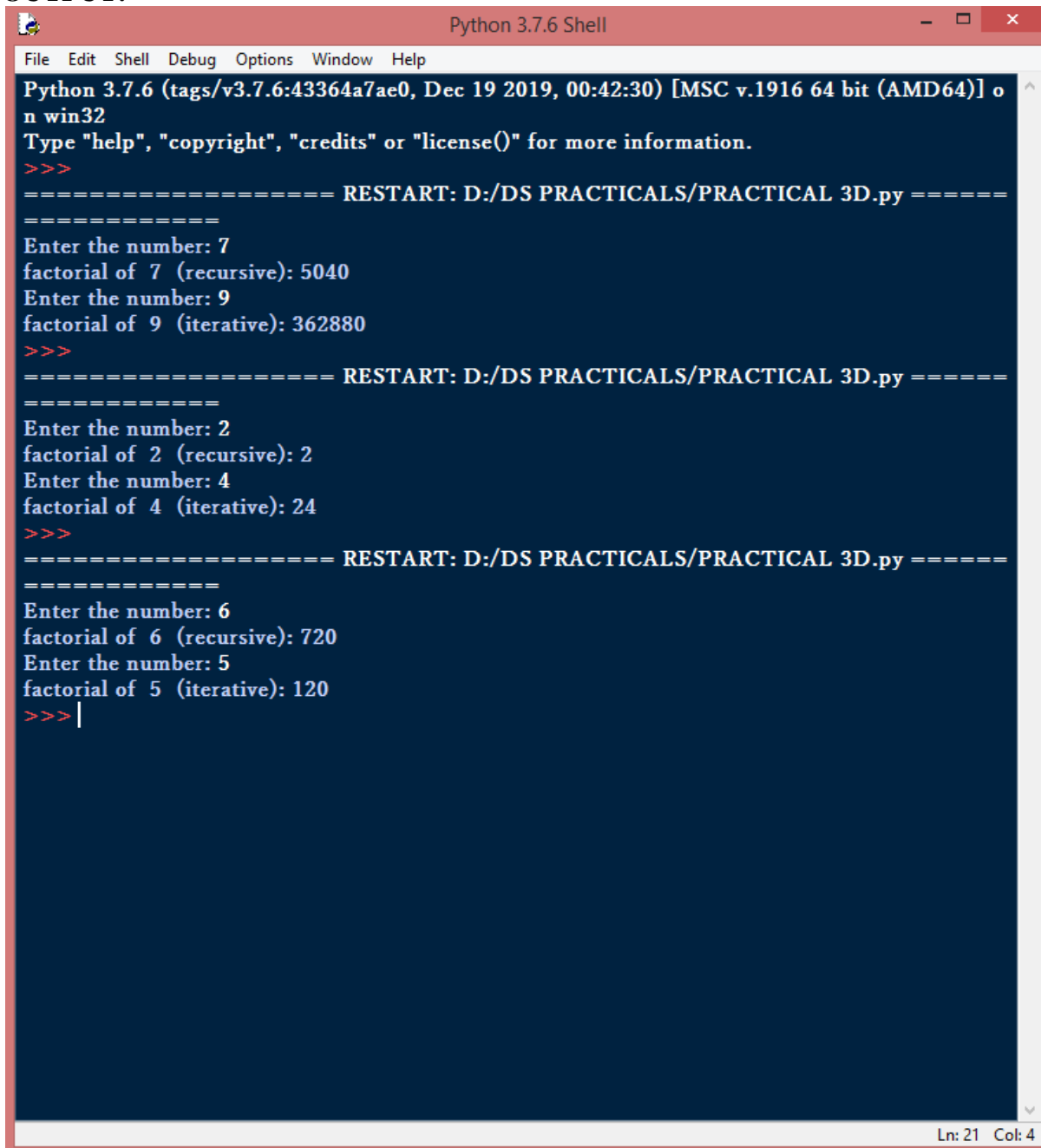
def iter_factorial(n):
    factorial=1
    if int(n) >= 1:
        for i in range (1,int(n)+1):
            factorial = factorial * i
        return factorial

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

print("factorial of ",num," (iterative): ",end="")
print(iter_factorial(num))
|
```

Ln: 25 Col: 0

OUTPUT:



```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 3D.py =====
=====
Enter the number: 7
factorial of 7 (recursive): 5040
Enter the number: 9
factorial of 9 (iterative): 362880
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 3D.py =====
=====
Enter the number: 2
factorial of 2 (recursive): 2
Enter the number: 4
factorial of 4 (iterative): 24
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 3D.py =====
=====
Enter the number: 6
factorial of 6 (recursive): 720
Enter the number: 5
factorial of 5 (iterative): 120
>>> |
```

Ln: 21 Col: 4

PRACTICAL NO:-4

AIM:-4 Perform Queues Operations using Circular Array Implementation.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%204.py>

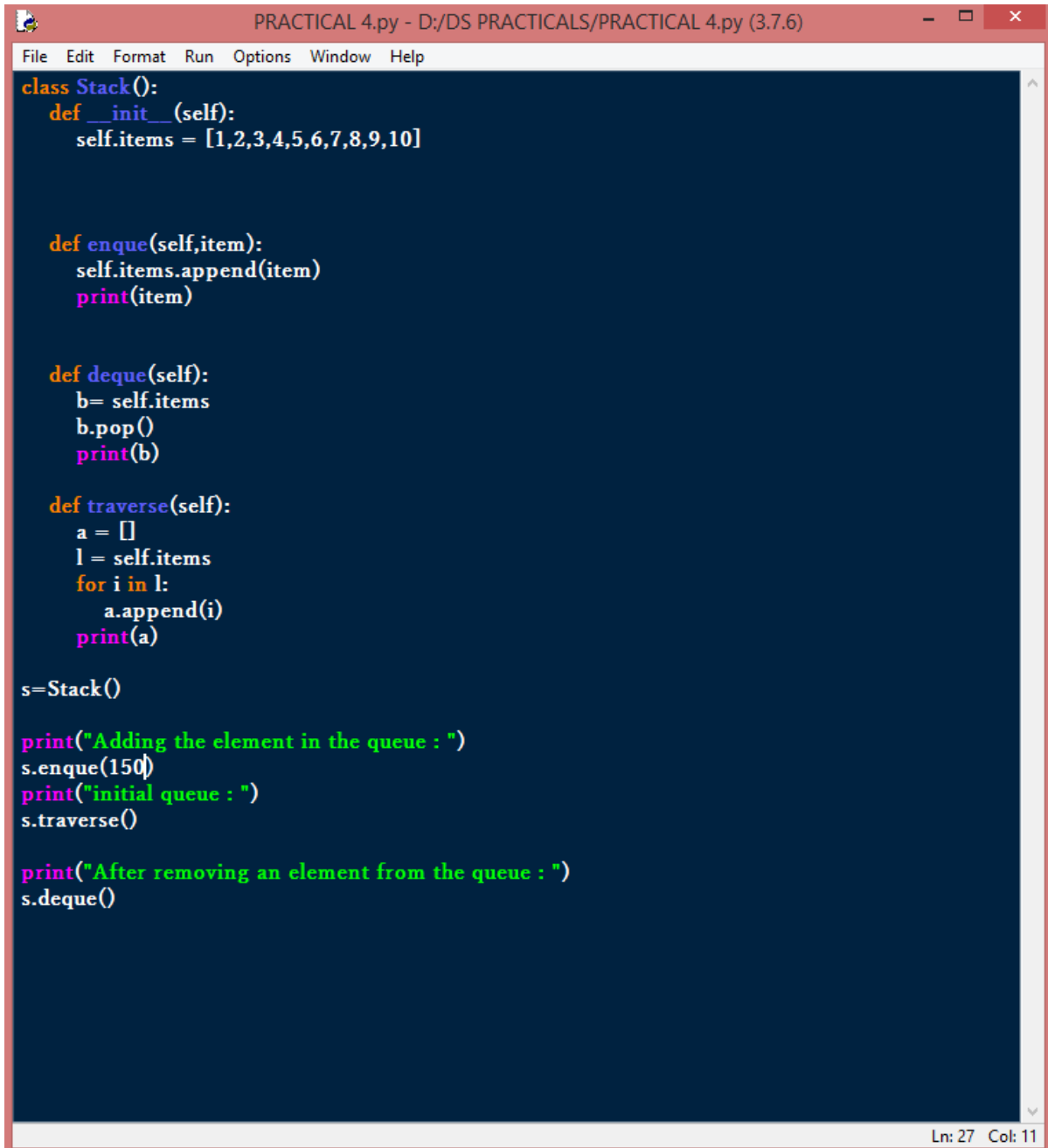
Theory:

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 had 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:



```
class Stack():
    def __init__(self):
        self.items = [1,2,3,4,5,6,7,8,9,10]

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

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

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

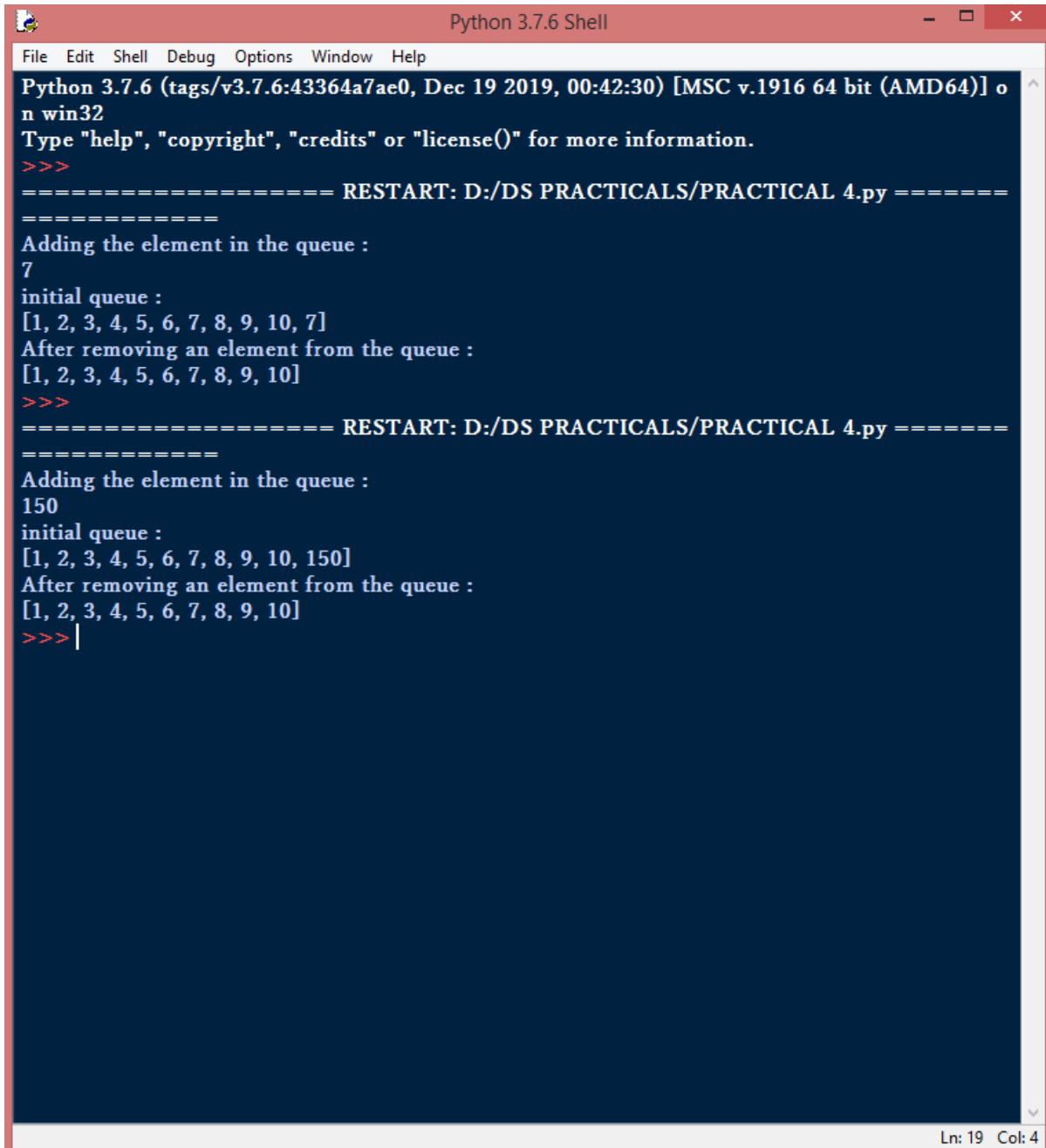
s=Stack()

print("Adding the element in the queue : ")
s.enqueue(150)
print("initial queue : ")
s.traverse()

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

Ln: 27 Col: 11

OUTPUT:



```
Python 3.7.6 (tags/v3.7.6:43364a7ac0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 4.py =====
=====
Adding the element in the queue :
7
initial queue :
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 7]
After removing an element from the queue :
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 4.py =====
=====
Adding the element in the queue :
150
initial queue :
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 150]
After removing an element from the queue :
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> |
```

Ln: 19 Col: 4

PRACTICAL NO:-5

AIM:-5 Write a program to search an element from a given list. Give the user option to perform Liner or Binary Search.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%205.py>

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:

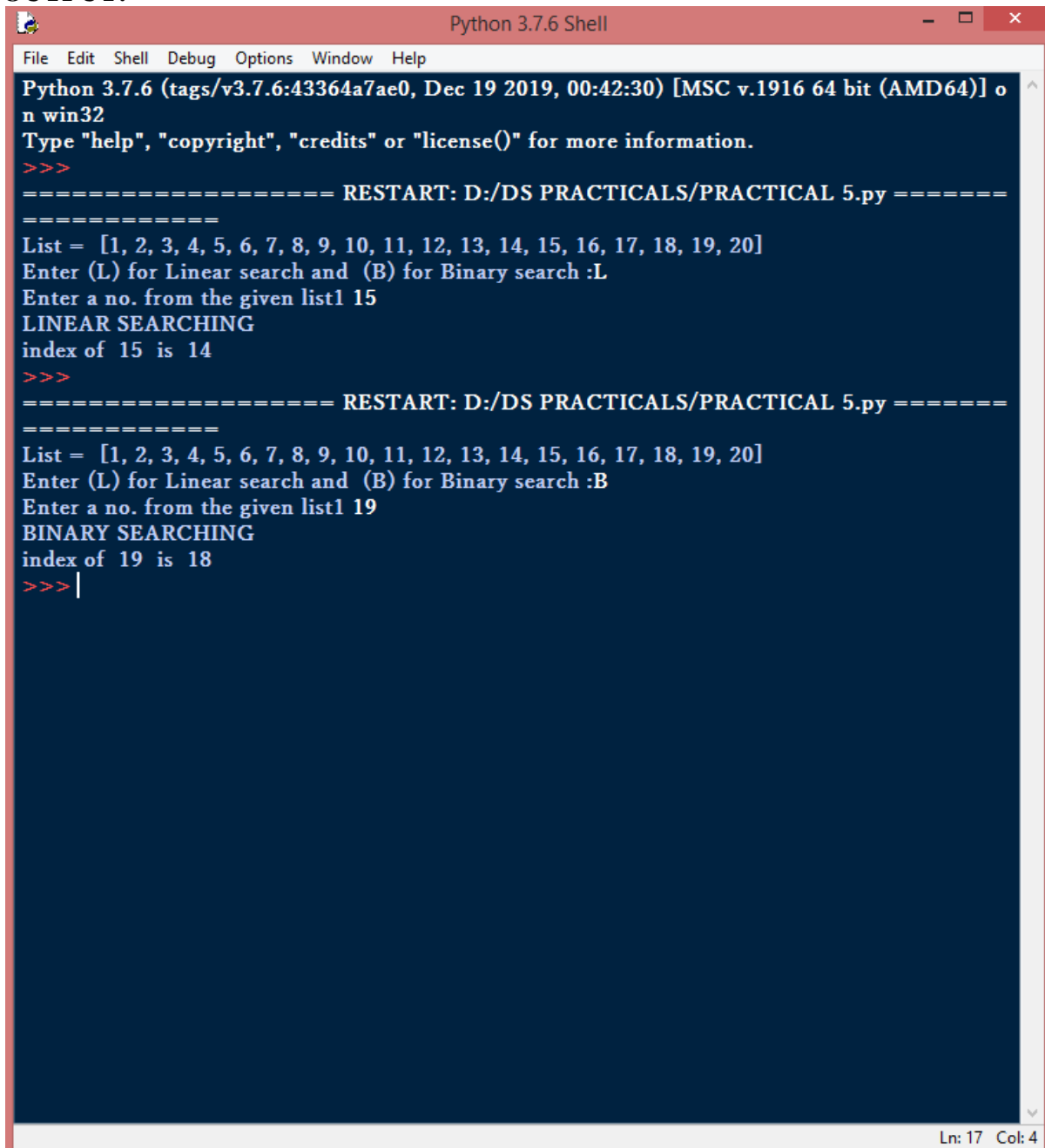
```
PRACTICAL 5.py - D:/DS PRACTICALS/PRACTICAL 5.py (3.7.6)
File Edit Format Run Options Window Help
list1 = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
print("List = ",list1)
size = len(list1)
def binary_search(x):
    print("BINARY SEARCHING")
    low = 0
    high = len(list1) - 1
    mid = 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")

Ln: 13 Col: 0
```

OUTPUT:



```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ac0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 5.py =====
=====
List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Enter (L) for Linear search and (B) for Binary search :L
Enter a no. from the given list 15
LINEAR SEARCHING
index of 15 is 14
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 5.py =====
=====
List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Enter (L) for Linear search and (B) for Binary search :B
Enter a no. from the given list 19
BINARY SEARCHING
index of 19 is 18
>>> |
```

Ln: 17 Col: 4

PRACTICAL NO:-6

AIM:-6 WAP to sort a list of Elements. Give the user option to perform sorting using Insertion sort, Bubble sort or Selection sort.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%206.py>

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:

```
PRACTICAL 6.py - D:/DS PRACTICALS/PRACTICAL 6.py (3.7.6)
File Edit Format Run Options Window Help
list1 = [45,52,78,35,69,87,12,6,8,17,96]
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 Selection Sort and (I) for Insertion Sort %n Enter")
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")

Ln: 28 Col: 22
```

OUTPUT:

```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 6.py =====
=====
List = [45, 52, 78, 35, 69, 87, 12, 6, 8, 17, 96]
Enter (B) for Bubble Sort, (S) for elsection Sort and (I) for Insertion Sort
Enter here:B
Bubble Sorting
[6, 8, 12, 17, 35, 45, 52, 69, 78, 87, 96]
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 6.py =====
=====
List = [45, 52, 78, 35, 69, 87, 12, 6, 8, 17, 96]
Enter (B) for Bubble Sort, (S) for elsection Sort and (I) for Insertion Sort
Enter here:S
Selection Sorting
[6, 8, 12, 17, 35, 45, 52, 69, 78, 87, 96]
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 6.py =====
=====
List = [45, 52, 78, 35, 69, 87, 12, 6, 8, 17, 96]
Enter (B) for Bubble Sort, (S) for elsection Sort and (I) for Insertion Sort
Enter here:I
Insertion Sorting
[6, 8, 12, 17, 35, 45, 52, 69, 78, 87, 96]
>>> |
```

Ln: 24 Col: 4

PRACTICAL NO:-7A

AIM:-7a) Write a Program to Implement the Collision Technique.

LINK:<https://github.com/codewithkk/DS/blob/master/TestPractical%207a.py>

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:

```
PRACTICAL 7A.py - D:/DS PRACTICALS/PRACTICAL 7A.py (3.7.6)
File Edit Format Run Options Window Help
size_list=int(input("Enter the size of list:"))

def search_from_hash(key,hash_list):
    searched_index=hash_function(key)
    if hash_list[searched_index]:
        print("value found")
    else:
        print("Value not in list")

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

def map_hash2index(hash_return_value):
    return hash_return_value

def create_hash_table(list_values,main_list):
    for value in list_values:
        hash_return_value=hash_function(value)
        list_index=map_hash2index(hash_return_value)
        if main_list[list_index]:
            print("collision detected")
        else:
            main_list[list_index]=value

list_values = [1,3,5,7,9,13,16,78,77,998]

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

search_from_hash(30,main_list)
```

Ln: 25 Col: 0

OUTPUT:

```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 7A.py =====
=====
Enter the size of list:5
[None, None, None, None, None]
collision detected
collision detected
collision detected
collision detected
collision detected
[5, 1, 7, 3, 9]
value found
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 7A.py =====
=====
Enter the size of list:10
[None, None, None, None, None, None, None, None, None, None]
collision detected
collision detected
collision detected
[None, 1, None, 3, None, 5, 16, 7, 78, 9]
Value not in list
>>> |
```

Ln: 23 Col: 4

PRACTICAL NO:-7B

AIM:-7b) Write a program to implement the concept of liner Probing.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%207b.py>

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:

```
PRACTICAL 7B.py - D:/DS PRACTICALS/PRACTICAL 7B.py (3.7.6)
File Edit Format Run Options Window Help
class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys, lowerrange, higherrange)

    def get_key_value(self):
        return self.value

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

if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [23, 43, 1, 87]
    list_of_list_index = [None, None, None, None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        # print(Hash(value, 0, len(list_of_keys)).get_key_value())
        list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
        print("hash value for " + str(value) + " is : " + str(list_index))
        if list_of_list_index[list_index]:
            print("Collision detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index) - 1:
                    list_index = 0
                else:
                    list_index += 1
            list_full = False
            while list_of_list_index[list_index]:
                if list_index == old_list_index:
                    list_full = True
                    break
                if list_index + 1 == len(list_of_list_index):
                    list_index = 0
                else:
                    list_index += 1
            if list_full:
                print("List was full . Could not save")
            else:
```

Ln: 47 Col: 0

```
PRACTICAL 7B.py - D:/DS PRACTICALS/PRACTICAL 7B.py (3.7.6)
File Edit Format Run Options Window Help

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

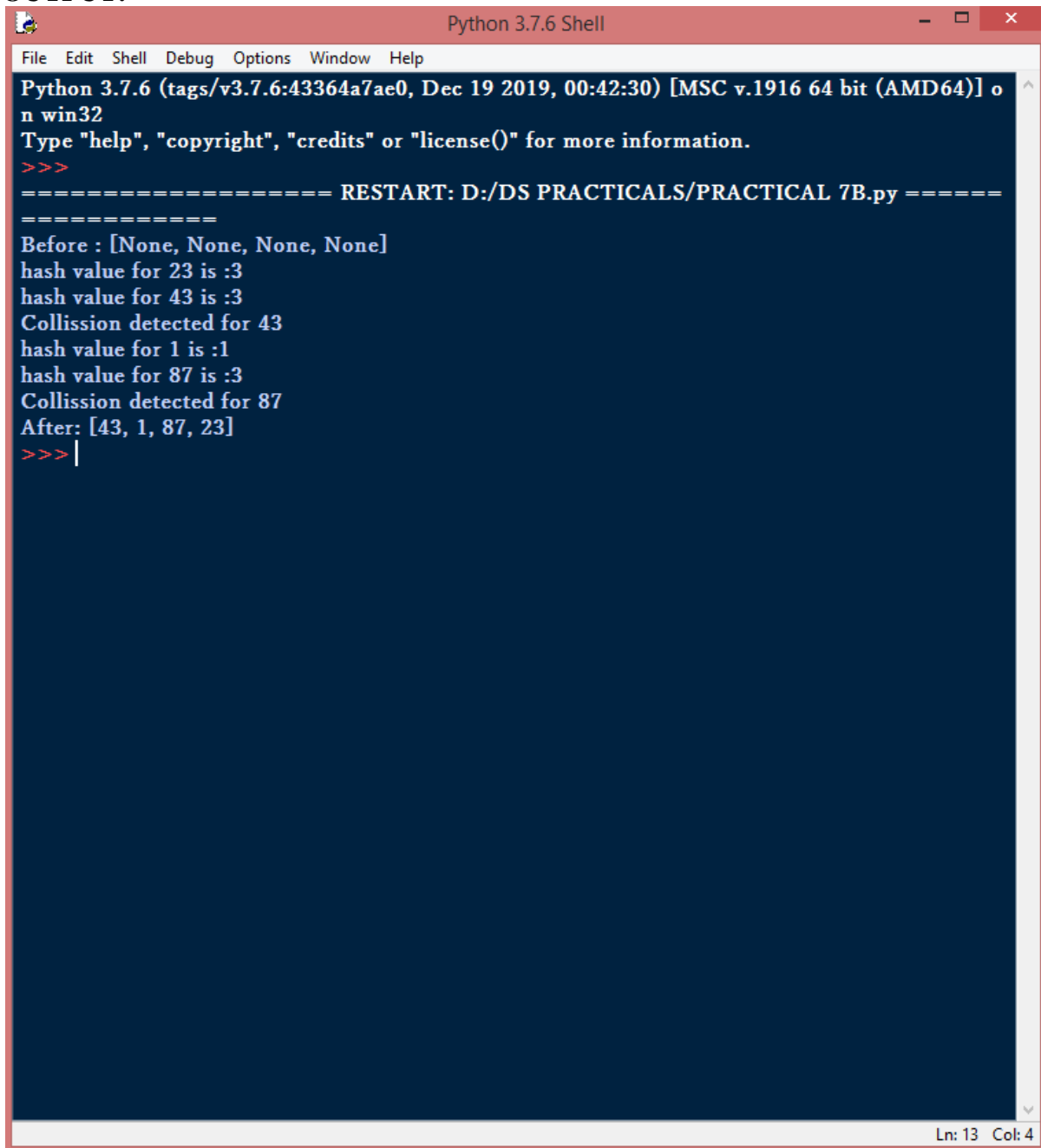
if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [23,43,1,87]
    list_of_list_index = [None,None,None,None]
    print("Before: " + str(list_of_list_index))
    for value in list_of_keys:
        #print(Hash(value,0,len(list_of_keys)).get_key_value())
        list_index = Hash(value,0,len(list_of_keys)).get_key_value()
        print("hash value for " + str(value) + " is:" + str(list_index))
        if list_of_list_index[list_index]:
            print("Collision detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index)-1:
                    list_index = 0
                else:
                    list_index += 1
            list_full = False
            while list_of_list_index[list_index]:
                if list_index == old_list_index:
                    list_full = True
                    break
                if list_index+1 == len(list_of_list_index):
                    list_index = 0
                else:
                    list_index += 1
            if list_full:
                print("List was full . Could not save")
            else:
                list_of_list_index[list_index] = value

        else:
            list_of_list_index[list_index] = value

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

Ln: 47 Col: 0
```


OUTPUT:

A screenshot of a Python 3.7.6 Shell window. The window has a red title bar with the text "Python 3.7.6 Shell" and standard window controls. Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area is a dark blue terminal with white text. The output shows the start of a Python program, a restart message, and a list of hash values with collision detection for the numbers 43 and 87. The final state of a list is shown as [43, 1, 87, 23].

```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 7B.py =====
Before : [None, None, None, None]
hash value for 23 is :3
hash value for 43 is :3
Collision detected for 43
hash value for 1 is :1
hash value for 87 is :3
Collision detected for 87
After: [43, 1, 87, 23]
>>> |
```

Ln: 13 Col: 4

PRACTICAL NO:-8

AIM:-8 Write a Program for Inorder, Postorder and Preorder traversal of tree.

LINK:<https://github.com/codewithkk/DS/blob/master/Practical%208.py>

Theory:

Inorder Traversal:

For binary search trees (BST), Inorder Traversal specifies the nodes in non-descending order. In order to obtain nodes from BST in non-increasing order, a variation of inorder traversal may be used where inorder traversal is reversed.

Preorder Traversal:

Preorder traversal will create a copy of the tree. Preorder Traversal is also used to get the prefix expression of an expression.

Postorder Traversal:

Postorder traversal is used to get the postfix expression of an expression given

Inorder

Traverse the left sub-tree, (recursively call inorder(root -> left). Visit and print the root node.
Traverse the right sub-tree, (recursively call inorder(root -> right).

Preorder

Visit and print the root node. Traverse the left sub-tree, (recursively call inorder(root -> left).
Traverse the right sub-tree, (recursively call inorder(root -> right).

Postorder

Traverse the left sub-tree, (recursively call inorder(root -> left). Traverse the right sub-tree, (recursively call inorder(root -> right). Visit and print the root node.

CODE:



```
import random

random.seed(23)

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

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

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

    return root

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

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

Ln: 62 Col: 0

```
PRACTICAL 8.py - D:/DS PRACTICALS/PRACTICAL 8.py (3.7.6)
File Edit Format Run Options Window Help

    return root

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

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

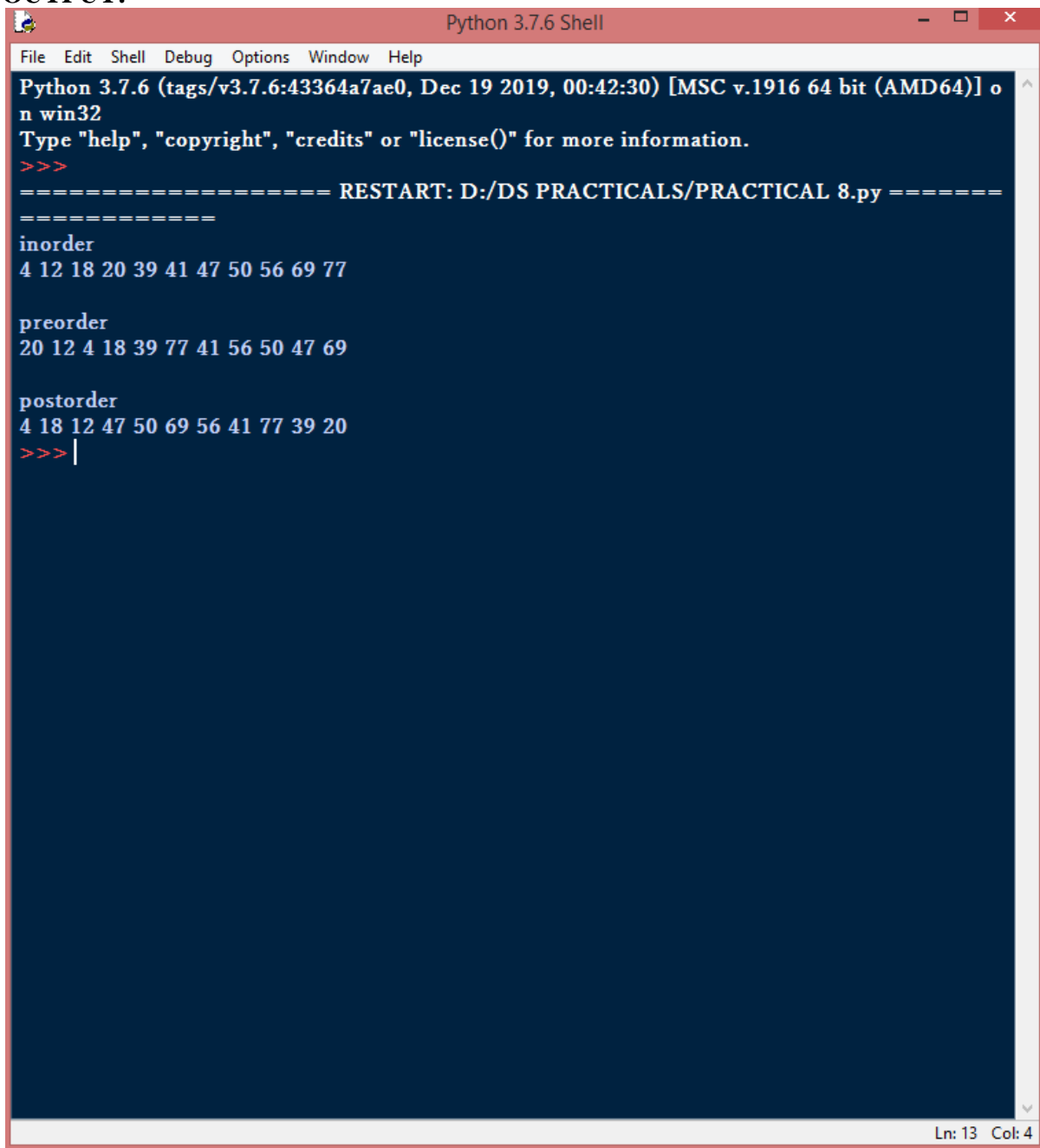
def printPostorder(root):
    if root:
        printPostorder(root.leftChild)
        printPostorder(root.rightChild)
        print(root.val, end=" ")

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

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

Ln: 62 Col: 0

OUTPUT:



```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/DS PRACTICALS/PRACTICAL 8.py =====
>>>
inorder
4 12 18 20 39 41 47 50 56 69 77

preorder
20 12 4 18 39 77 41 56 50 47 69

postorder
4 18 12 47 50 69 56 41 77 39 20
>>> |
```

Ln: 13 Col: 4