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. Krutik Raju Dhamecha

Roll No : 312 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: 312

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.	

Practical1.Implement the following for Array

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 :Storing Data in Arrays. Assigning values to an elementin an array is similar to assigning values to scalar variables. Simply reference an individual element of anarray using the array name and the index inside parentheses, then use the assignment operator (=) followed by a value.

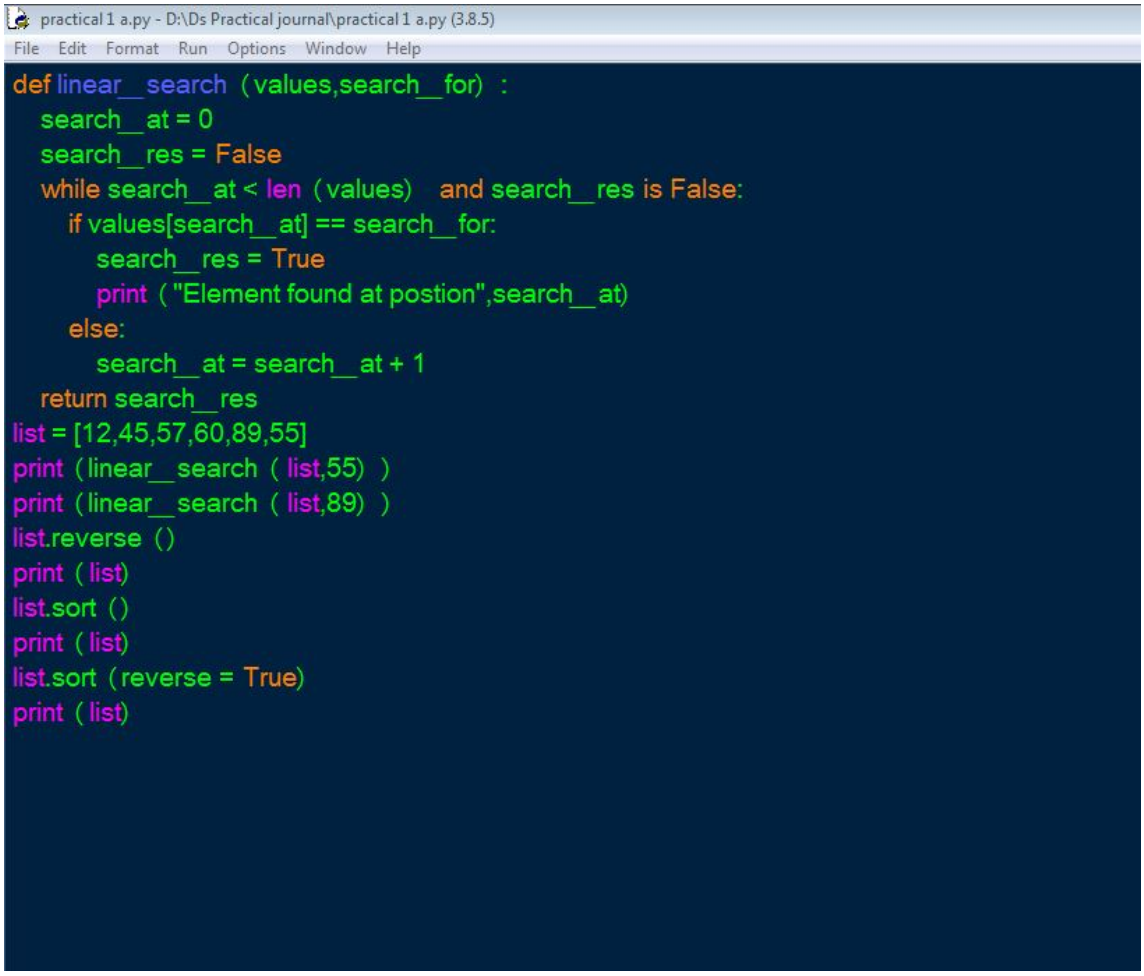
Following are the basic operations supported by an array.

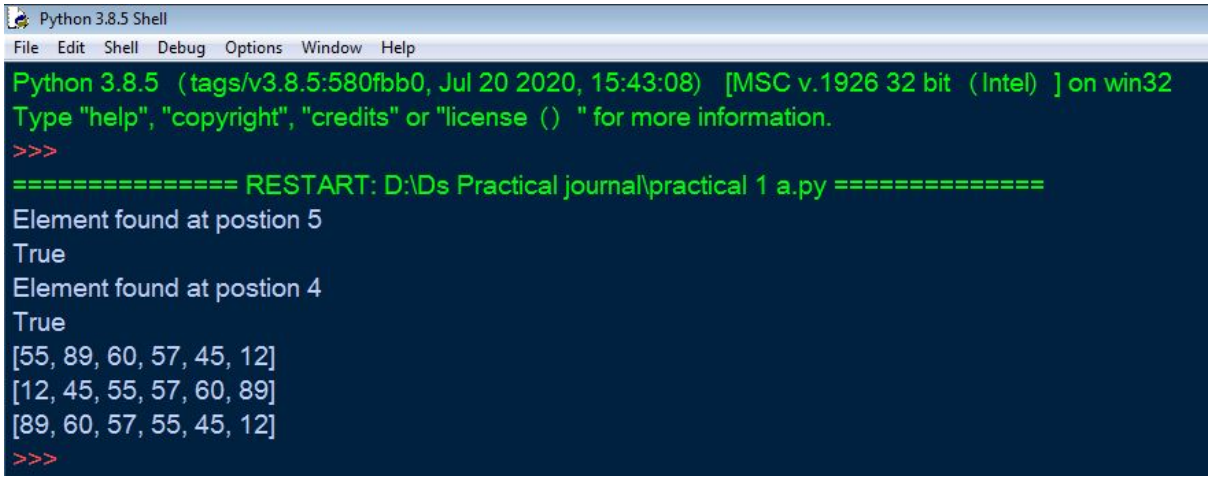
Traverse − print all the array elements one by one.

Insertion − Adds an element at the given index.

Deletion − Deletes an element at the given index.

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





Practical no.1b

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

Theory:add() − add elements of two matrices.

subtract() − subtract elements of two matrices.

divide() − divide elements of two matrices.

multiply() − multiply elements of two matrices.

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

sqrt() − square root of each element of matrix.

sum(x,axis) − add to all the elements in matrix. Second argument is optional, it is used when we want to compute the column sum if axis is 0 and row sum if axis is 1.

“T” − It performs transpose of the specified matrix.

Pratical 1b.py - D:\Ds Practical journal\Pratical 1b.py (3.8.5)

File Edit Format Run Options Window Help

```
print ( "Matrix operation")
print ( "1.Addition")
print ( "2.Mutiplication")
print ( "3.Transpose")
x = [[12,3,21],[2,43,5],[5,32,53]]
y = [[11,4,6],[21,7,9],[4,8,43]]
result = [[0,0,0],[0,0,0],[0,0,0]]
ch = int ( input ( "Enter your choice:") )
if ch == 1:
   for i in range ( len ( x) ) :
      for j in range ( len ( y[0]) ) :
         result[i][j] = x[i][j] + y[i][j]
   for r in result:
      print ( r)
elif ch == 2:
   for i in range ( len ( x) ) :
      for j in range ( len ( y[0]) ) :
         for k in range  ( len ( y) ) :
            result[i][j] = x[i][j] * y[i][j]
   for r in result:
      print ( r)
elif ch == 3:
   for i in range ( len ( x) ) :
      for j in range ( len ( y[0]) ) :
         result[j][i] = x[i][j]
   for r in result:
      print ( r)
else:
   print ( "Invalid choice")
```

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 ( tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08)  [MSC v.1926 32 bit ( Intel) ] on win32
Type "help", "copyright", "credits" or "license ()  " for more information.
>>>
================ RESTART: D:\Ds Practical journal\Pratical 1b.py ===============
Matrix operation
1.Addition
2.Mutiplication
3.Transpose
Enter your choice:1
[23, 7, 27]
[23, 50, 14]
[9, 40, 96]
>>>
```

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.

Insertion in a Linked List

Inserting element in the linked list involves reassigning the pointers from the existing nodes to the newly inserted node. Depending on whether the new data element is getting inserted at the beginning or at the middle or at the end of the linked list.

Deleting an Item form a Linked List

We can remove an existing node using the key for that node. In the below program we locate the previous node of the node which is to be deleted. Then point the next pointer of this node to the next node of the node to be deleted.

Searching in linked list

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

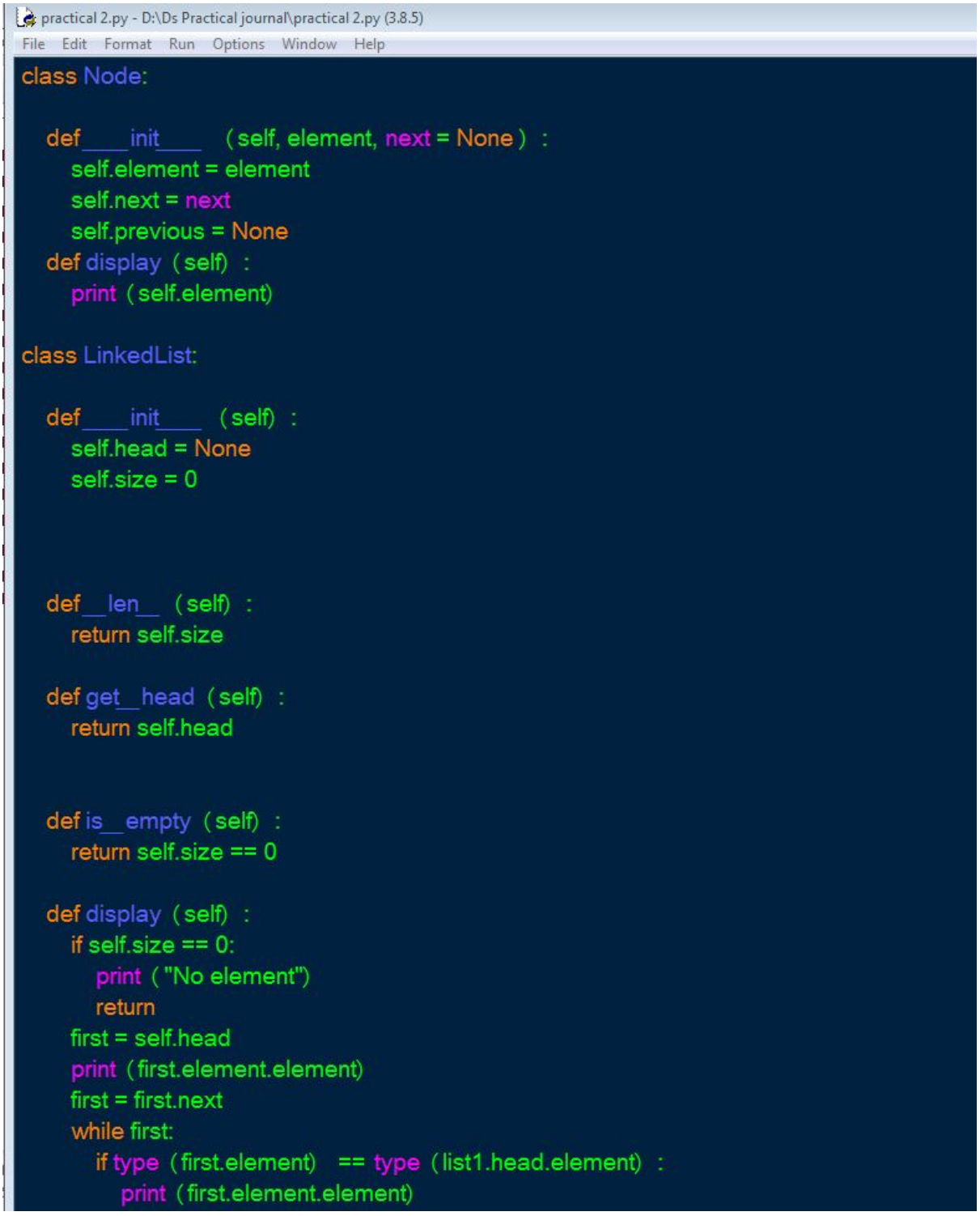
Reversing a Linked List

To reverse a LinkedList recursively we need to divide the LinkedList into two parts: head and remaining. Head points to the first element initially. Remaining points to the next element from the head. We traverse theLinkedList recursively until the second last

element.

Concatenating Linked Lists

Concatenate the two lists by traversing the first list until we reach it's a tail node and then point the next of the tail node to the head node of the second list. Store this concatenated list in the first list.



```
            first = first.next
        print (first.element)
        first = first.next

def reverse_display (self) :
    if self.size == 0:
        print ("No element")
        return None
    last = list1.get_tail ()
    print (last.element)
    while last.previous:
        if type (last.previous.element) == type (list1.head) :
            print (last.previous.element.element)
            if last.previous == self.head:
                return None
            else:
                last = last.previous
        print (last.previous.element)
        last = last.previous



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

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


def remove_head (self) :
    if self.is_empty () :
```

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

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

def find_second_last_element (self) :



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


    else:
        print ("Size not sufficient")

    return None



def remove_tail (self) :
    if self.is_empty () :
```

```
        print ("Empty Singly linked list")
    elif self.size == 1:
        self.head == None
        self.size -= 1
    else:
        Node = self.find__second__last__element ()
        if Node:
            Node.next = None
            self.size -= 1

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

def get__previous__node__at (self,index) :
    if index == 0:
        print ('No previous value')
        return None
    return list1.get__node__at (index) .previous

def remove__between__list (self,position) :
    if position > self.size-1:
        print ("Index out of bound")
    elif position == self.size-1:
        self.remove__tail ()
    elif position == 0:
```

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

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

def search(self,search_value):
    index = 0
    while(index < self.size):
        value = self.get_node_at(index)
        if type(value.element) == type(list1.head):
            print("Searching at " + str(index) + " and value is " + str(value.element.element))
        else:
            print("Searching at " + str(index) + " and value is " + str(value.element))
        if value.element == search_value:
            print("Found value at " + str(index) + " location")
```

practical 2.py - D:\Ds Practical journal\practical 2.py (3.8.5)

File Edit Format Run Options Window Help

```
            return True
        index += 1
    print ("Not Found")
    return False

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

    else:
        self.head = linkedlist_value.head
        self.size = linkedlist_value.size

l1 = Node ('element 1')
list1 = LinkedList ()
list1.add_head (l1)
list1.add_tail ('element 2')
list1.add_tail ('element 3')
list1.add_tail ('element 4')
list1.get_head () .element.element
list1.add_between_list (2, 'element between')
list1.remove_between_list (2)

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

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license () " for more information.
>>>
================ RESTART: D:\Ds Practical journal\practical 2.py ================
element 8
element 7
element 6
element 5
element 4
element 3
element 2
element 1
Searching at 0 and value is element 1
Searching at 1 and value is element 2
Searching at 2 and value is element 3
Searching at 3 and value is element 4
Searching at 4 and value is element 5
Searching at 5 and value is element 6
Found value at 5 location
>>>
```

Practical no. 3: Implement the following for Stack

Aim: Perform Stack operations using Array implementation

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

There are two types of operations in Stack-

Push– To add data into the stack.

Pop– To remove data from the stack

Practical 3a.py - D:\Ds Practical journal\Practical 3a.py (3.8.5)

File Edit Format Run Options Window Help

```
class Stack:

    def __init__ (self) :
        self.stack_arr = []

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

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

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

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

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

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license () " for more information.
>>>
========== RESTART: D:\Ds Practical journal\Practical 3a.py ==========
[4, 5]
>>>
```

Practical no.3b

Aim: Implement Tower of Hanoi.

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

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

Recursion

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

Iteration

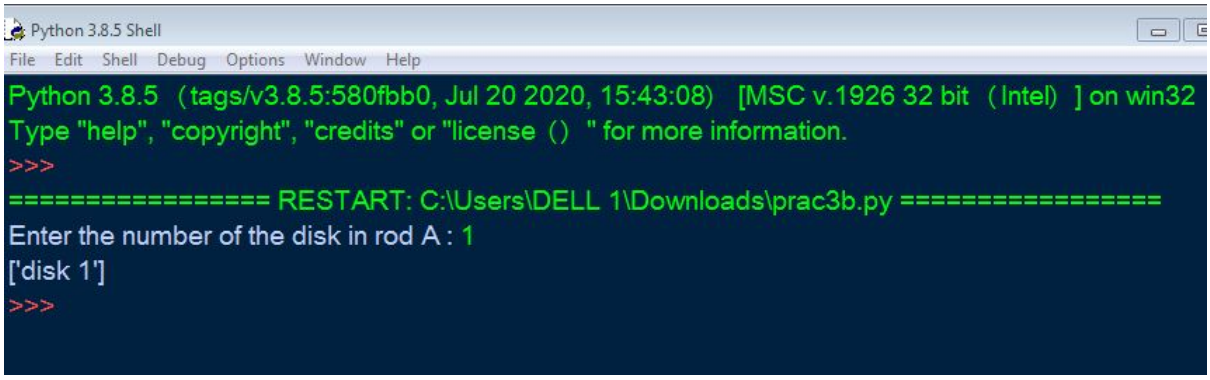
Repeating identical or similar tasks without making errors is something that computers do well and people do poorly. Repeated execution of a set of statements is called iteration.
Because iteration is so common,Python provides several language features to make it easier.

p3b.py - D:/Ds Practical journal/p3b.py (3.8.5)

File Edit Format Run Options Window Help

```
class Stack:

    def __init__ (self) :
        self.stack__arr = []

    def push (self,value) :
        self.stack__arr.append (value)

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

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

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

A = Stack ()
B = Stack ()
C = Stack ()
def Hanoi (n, fromrod,to,temp) :
    if n == 1:
        fromrod.pop ()
        to.push ('disk 1')
```

```
        to.push ('disk 1')
        if to.display () != None:
            print (to.display () )

    else:

        Hanoi (n-1, fromrod, temp, to)
        fromrod.pop ()
        to.push (f'disk {n} ')
        if to.display () != None:
            print (to.display () )
        Hanoi (n-1, temp, to, fromrod)

n = int ( input ('Enter the number of the disk in rod A : ') )
for i in range (n) :
    A.push (f'disk {i+1} ')

Hanoi (n, A, C, B)
```



Practical:3c

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

Theory:Polynomial is a mathematical expression that consists of variables and coefficients. for example x^2 – 4x +7

In the Polynomial linked list, the coefficients and exponents of the polynomial are defined as the data node of the list.

For adding two polynomials that are stored as a linked list. We need to add the coefficients of variables with the same power. In a linked list node contains 3 members, coefficient value link to the next node.

a linked list that is used to store Polynomial looks like −

Polynomial : 4x7 + 12x2 + 45

Practical 3c.py - D:\Ds Practical journal\Practical 3c.py (3.8.5)

File Edit Format Run Options Window Help

```
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:
```

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

def reverse__display (self) :
    if self.size == 0:
        print ("No element")
        return None
    last = my__list.get__tail ()
    print (last.element)
    while last.previous:
        if type (last.previous.element)  == type (my__list.head) :
            print (last.previous.element.element)
            if last.previous == self.head:
                return None
            else:
                last = last.previous
        print (last.previous.element)
        last = last.previous



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

def get__tail (self) :
```

```
    last__object = self.head
    while (last__object.next != None) :
        last__object = last__object.next
    return last__object


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

def add__tail (self,e) :
    new__value = Node (e)
    new__value.previous = self.get__tail ()
    self.get__tail () .next = new__value
    self.size += 1

def find__second__last__element (self) :
    #second__last__element = None


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


    else:
```

```
        print ("Size not sufficient")

    return None



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

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

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

```
    return my__list.get__node__at (index) .previous

def remove__between__list (self,position) :
    if position > self.size-1:
        print ("Index out of bound")
    elif position == self.size-1:
        self.remove__tail ()
    elif position == 0:
        self.remove__head ()
    else:
        prev__node = self.get__node__at (position-1)
        next__node = self.get__node__at (position+1)
        prev__node.next = next__node
        next__node.previous = prev__node
        self.size -= 1

def add__between__list (self,position,element) :
    element__node = Node (element)
    if position > self.size:
        print ("Index out of bound")
    elif position == self.size:
        self.add__tail (element)
    elif position == 0:
        self.add__head (element)
    else:
        prev__node = self.get__node__at (position-1)
        current__node = self.get__node__at (position)
        prev__node.next = element__node
        element__node.previous = prev__node
        element__node.next = current__node
        current__node.previous = element__node
        self.size += 1

def search (self,search__value) :
    index = 0
    while (index < self.size) :
```

```
            value = self.get__node__at (index)
            if value.element == search__value:
                return value.element
            index += 1
        print ("Not Found")
        return False

    def merge (self,linkedlist__value) :
        if self.size > 0:
            last__node = self.get__node__at (self.size-1)
            last__node.next = linkedlist__value.head
            linkedlist__value.head.previous = last__node
            self.size = self.size + linkedlist__value.size

        else:
            self.head = linkedlist__value.head
            self.size = linkedlist__value.size



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

my__list2 = LinkedList ()
my__list2.add__head (Node ( int (input (f"Enter coefficient for power {order} : ") ) ) )
for i in reversed (range (order) ) :
    my__list2.add__tail ( int (input (f"Enter coefficient for power {i} : ") ) )

for i in range (order + 1) :
    print (my__list.get__node__at (i) .element + my__list2.get__node__at (i) .element)
```

```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:\Ds Practical journal\Practical 3c.py ==========
Enter the order for polynomial : 4
Enter coefficient for power 4 : 4
Enter coefficient for power 3 : 7
Enter coefficient for power 2 : 2
Enter coefficient for power 1 : 4
Enter coefficient for power 0 : 5
Enter coefficient for power 4 : 5
Enter coefficient for power 3 : 5
Enter coefficient for power 2 : 5
Enter coefficient for power 1 : 5
Enter coefficient for power 0 : 5
9
12
7
9
10
>>>
```

Practical no.3d

Aim:WAP to calculate factorial and to compute the factors of a given no.

using recursion, (ii) using iteration

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

Recursion

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

Iteration

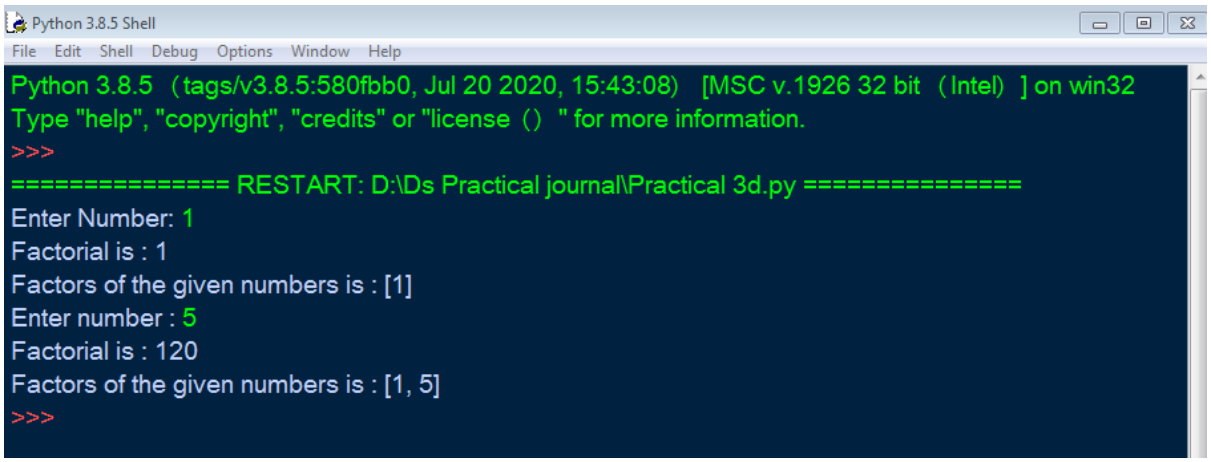
Repeating identical or similar tasks without making errors is something that computers do well and people do poorly. Repeated execution of a set of statements is called iteration.
Because iteration is so common,Python provides several language features to make it easier.

Practical 3d.py - D:\Ds Practical journal\Practical 3d.py (3.8.5)

File Edit Format Run Options Window Help

```
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)
```



Practical no.4

Aim: Perform Queues operations using Circular Array implementation

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

Circular Queue works by the process of circular increment i.e. when we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue.

Here, the circular increment is performed by modulo division with the queue size. That is,

if REAR + 1 == 5 (overflow!), REAR = (REAR + 1)%5 = 0 (start of queue)

The circular queue work as follows: two pointers FRONT and REAR FRONT track the first element of the queue REAR track the last elements of the queue initially, set value of FRONT and REARto -1

1.Enqueue Operation

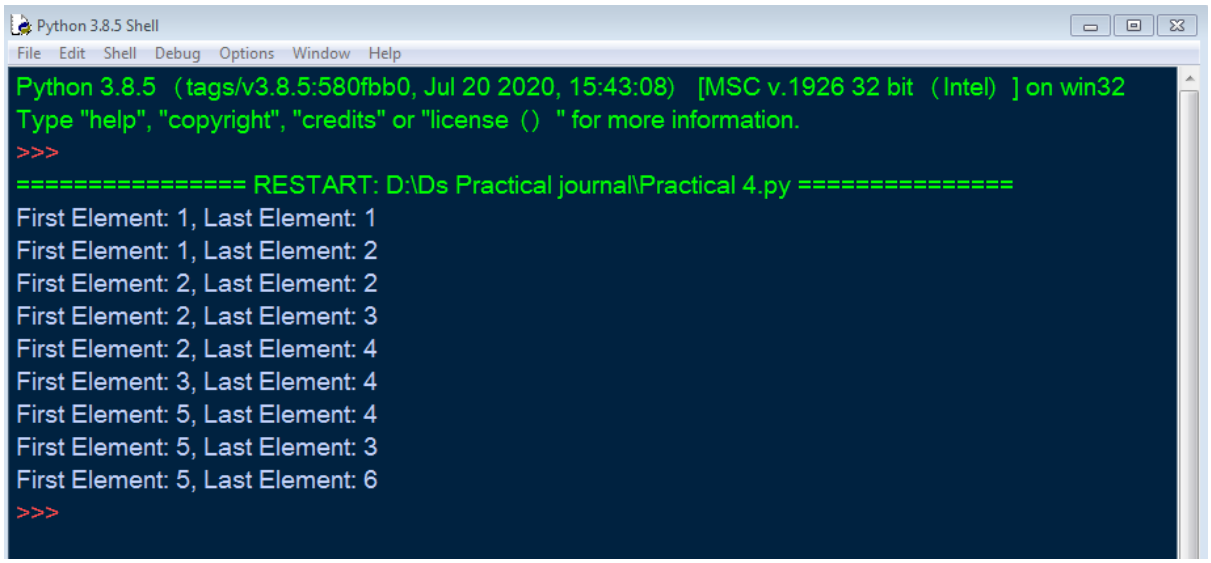
check if the queue is full for the first element, set value of FRONTto 0 circularly increase the REAR index by 1 (i.e. if the rear reaches the end, next it would be at the start of the queue) add the new element in the position pointed to by REAR

2. Dequeue Operation

check if the queue is empty return the value pointed by FRONT circularly increase the FRONT index by 1 for the last element, reset the values of FRONT and REAR to -1

```
        self.__front = (self.__front + 1) % len(self.__data)
        self.__size -= 1
        self.__back = (self.__front + self.__size - 1) % len(self.__data)
        return answer

    def dequeueEnd(self):
        """Remove and return the Last element of the queue.
        Raise Empty exception if the queue is empty.
        """
        if self.is__empty():
            raise Empty('Queue is empty')
        back = (self.__front + self.__size - 1) % len(self.__data)
        answer = self.__data[back]
        self.__data[back] = None        # help garbage collection
        self.__front = self.__front
        self.__size -= 1
        self.__back = (self.__front + self.__size - 1) % len(self.__data)
        return answer

    def enqueueEnd(self, e):
        """Add an element to the back of queue."""
        if self.__size == len(self.__data):
            self.__resize(2 * len(self.data))       # double the array size
        avail = (self.__front + self.__size) % len(self.__data)
        self.__data[avail] = e
        self.__size += 1
        self.__back = (self.__front + self.__size - 1) % len(self.__data)

    def enqueueStart(self, e):
        """Add an element to the start of queue."""
        if self.__size == len(self.__data):
            self.__resize(2 * len(self.__data))       # double the array size
        self.__front = (self.__front - 1) % len(self.__data)
        avail = (self.__front + self.__size) % len(self.__data)
        self.__data[self.__front] = e
        self.__size += 1
```

```
        self.__back = (self.__front + self.__size - 1) % len(self.__data)

    def __resize(self, cap):                # we assume cap >= len(self)
        """Resize to a new list of capacity >= len(self)."""
        old = self.__data                   # keep track of existing list
        self.__data = [None] * cap          # allocate list with new capacity
        walk = self.__front
        for k in range(self.__size):        # only consider existing elements
            self.__data[k] = old[walk]      # intentionally shift indices
            walk = (1 + walk) % len(old)    # use old size as modulus
        self.__front = 0                    # front has been realigned
        self.__back = (self.__front + self.__size - 1) % len(self.__data)

queue = ArrayQueue()
queue.enqueueEnd(1)
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
queue.__data
queue.enqueueEnd(2)
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
queue.__data
queue.dequeueStart()
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
queue.enqueueEnd(3)
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
queue.enqueueEnd(4)
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
queue.dequeueStart()
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
queue.enqueueStart(5)
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
queue.dequeueEnd()
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
queue.enqueueEnd(6)
print(f"First Element: {queue.__data[queue.__front]}, Last Element: {queue.__data[queue.__back]}")
```



Practical 5

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

Theory:Linear Search:

This linear search is a basic search algorithm whichsearches all the elements in the list and finds the required value. ... This is also known as sequential search.

Binary Search:

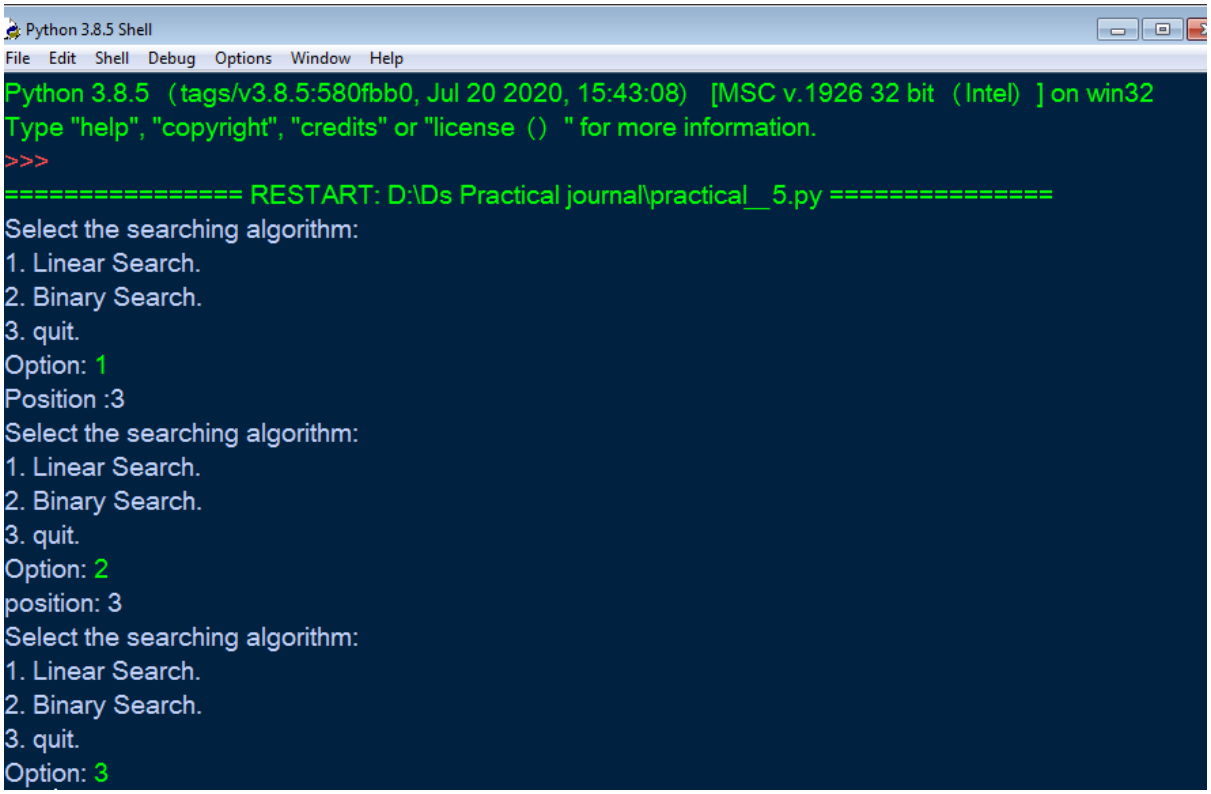
In computer science, a binary searchor half-interval search algorithm finds the position of a target value within a sorted array. The binary searchalgorithm can be classified as a dichotomies divide-and-conquer search algorithm and executes in logarithmic time.

practical_5.py - D:\Ds Practical journal\practical_5.py (3.8.5)

File Edit Format Run Options Window Help

```
class Array:

    def __init__ (self,array,number) :
        self.lst = sorted (array)
        self.number = number
    def binary_search (self,lst,n,start,end) :

        if start <= end:
            mid = (end + start) // 2
            if lst[mid] == n:

                return f'position: {mid} '
            elif lst[mid] > n:
                return binary_search (lst,n,start,mid-1)
            else:
                return binary_search (lst,n,mid + 1,end)
        else:
            return -1


    def linear_search (self,lst,n) :
        for i in range (len (lst) ) :
            if lst[i] == n:
                return f'Position : {i} '
        return -1

    def run_search (self) :
        while True:
            print ('Select the searching algorithm:')
            print ('1. Linear Search.')
            print ('2. Binary Search.')
            print ('3. quit.')
            opt = int (input ('Option: ') )
            if opt == 2:
                print (search.binary_search (self.lst,self.number,0, len (lst) -1) )
            elif opt == 1:
```

```
            print (search.linear__search (self.lst,self.number) )
        else:
            break


lst = [1,2,3,4,5,6,7,8]
number = 4
search = Array (lst,number)
search.run__search ()
```



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:Bubble Sort:

Bubble Sort is the simplest sorting algorithm that works by repeatedly

swapping the adjacent elements if they are in wrong order.

Selection Sort:

The selection sortalgorithm 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

Insertion Sort:

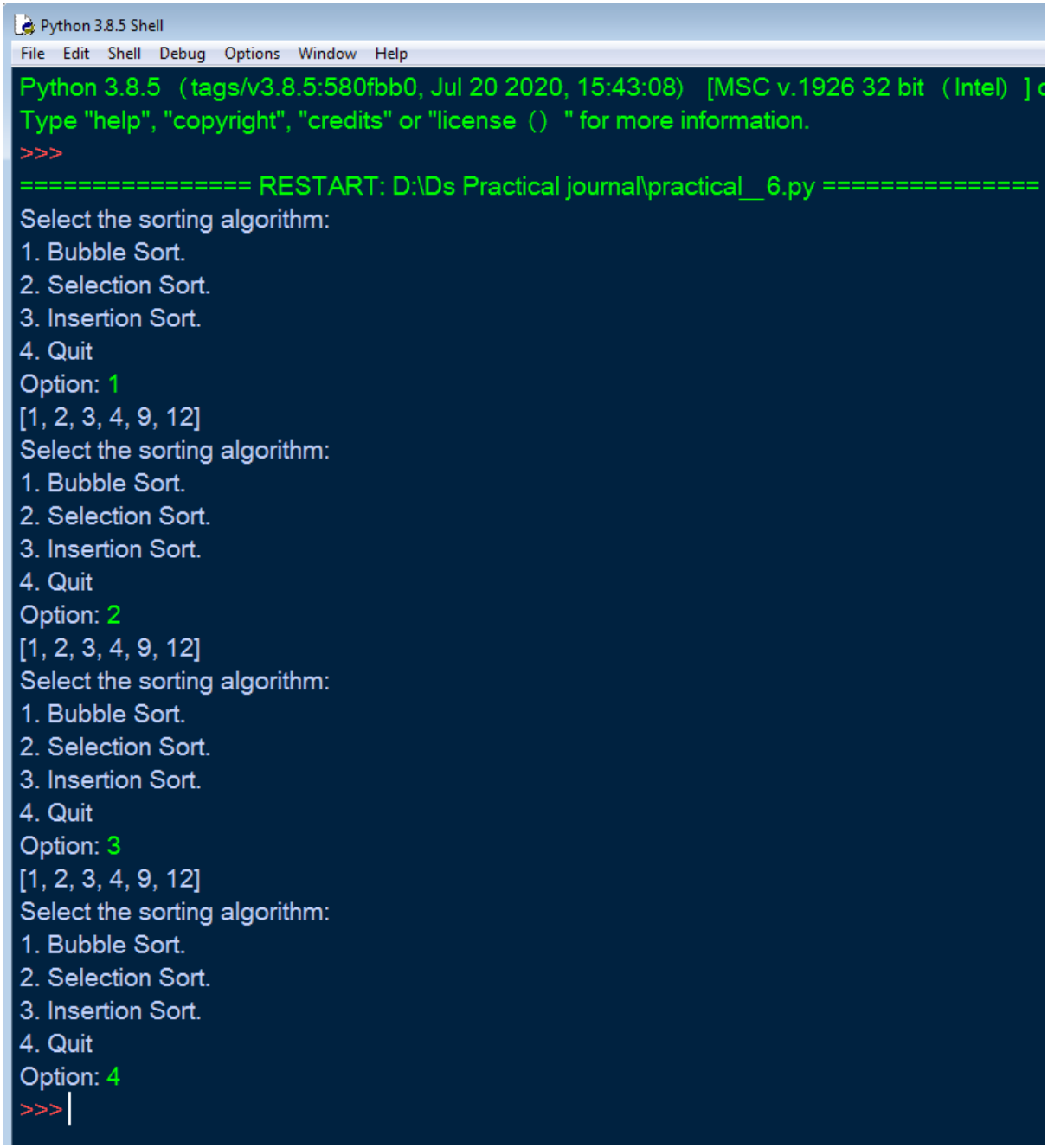
Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

practical_6.py - D:\Ds Practical journal\practical_6.py (3.8.5)

File Edit Format Run Options Window Help

```
class Sorting:

    def ____init____ (self,lst) :
        self.lst = lst

    def bubble__sort (self,lst) :
        for i in range (len (lst) ) :
            for j in range (len (lst) ) :
                if lst[i] < lst[j]:
                    lst[i],lst[j] = lst[j],lst[i]
                else:
                    pass
        return lst

    def selection__sort (self,lst) :
        for i in range (len (lst) ) :
            smallest__element = i
            for j in range (i+1,len (lst) ) :
                if lst[smallest__element] > lst[j]:
                    smallest__element = j
            lst[i],lst[smallest__element] = lst[smallest__element],lst[i]
        return lst

    def insertion__sort (self,lst) :
        for i in range (1, len (lst) ) :
            index = lst[i]
            j = i-1
            while j >= 0 and index < lst[j] :
                    lst[j + 1] = lst[j]
                    j -= 1
            lst[j + 1] = index
        return lst

    def run__sort (self) :
        while True:
            print ('Select the sorting algorithm:')
```

```
        print ('1. Bubble Sort.')
        print ('2. Selection Sort.')
        print ('3. Insertion Sort.')
        print ('4. Quit')
        opt = int (input ('Option: ') )
        if opt == 1:
            print (sort.bubble__sort (self.lst) )
        elif opt == 2:
            print (sort.selection__sort (self.lst) )
        elif opt == 3:
            print (sort.insertion__sort (self.lst) )
        else:
            break
lst = [4,2,3,9,12,1]
sort = Sorting (lst)
sort.run__sort ()
```



Practical 7 Implement the following for Hashing

Aim:Write a program to implement the collision technique.

Theory :Hashing:

Hashing is an important Data Structurewhich 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 thehash function

used.

Collisions:

A Hash Collision Attack is an attempt to find two input strings of a hash function that produce the same hashresult. If two separate inputs produce the same hashoutput, it is called a collision.

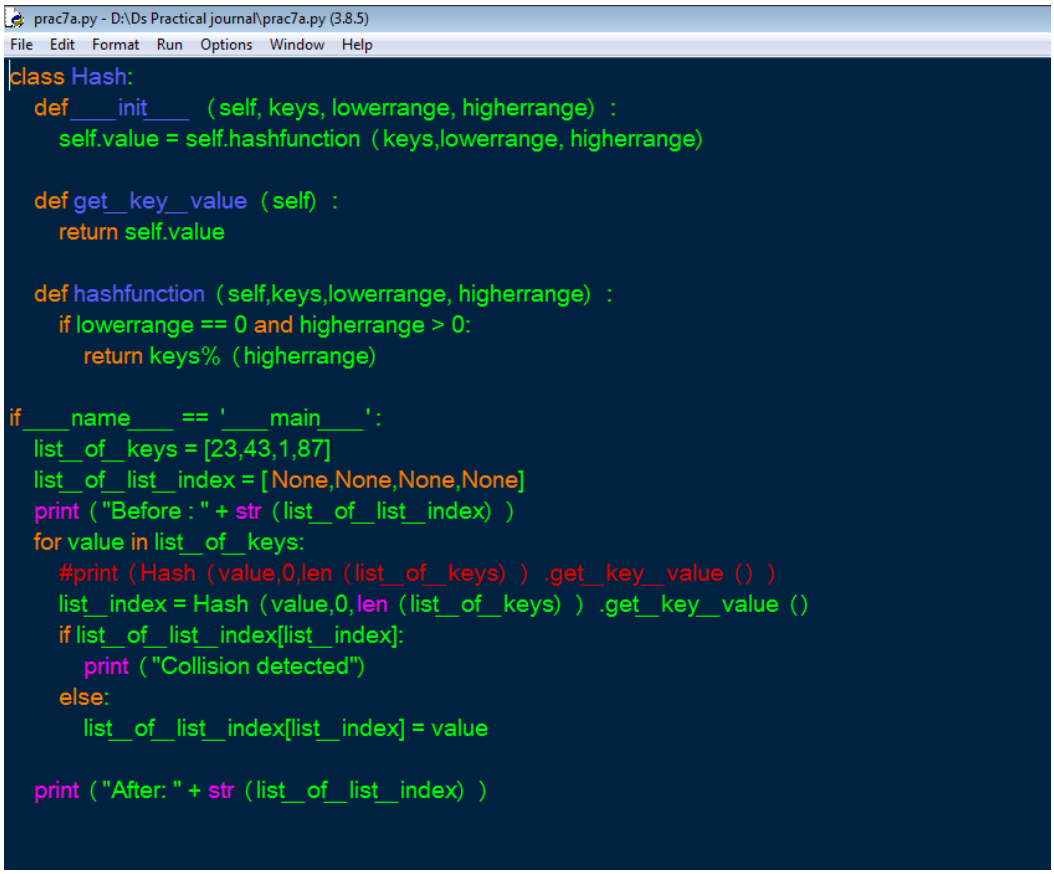
Collision Techniques:

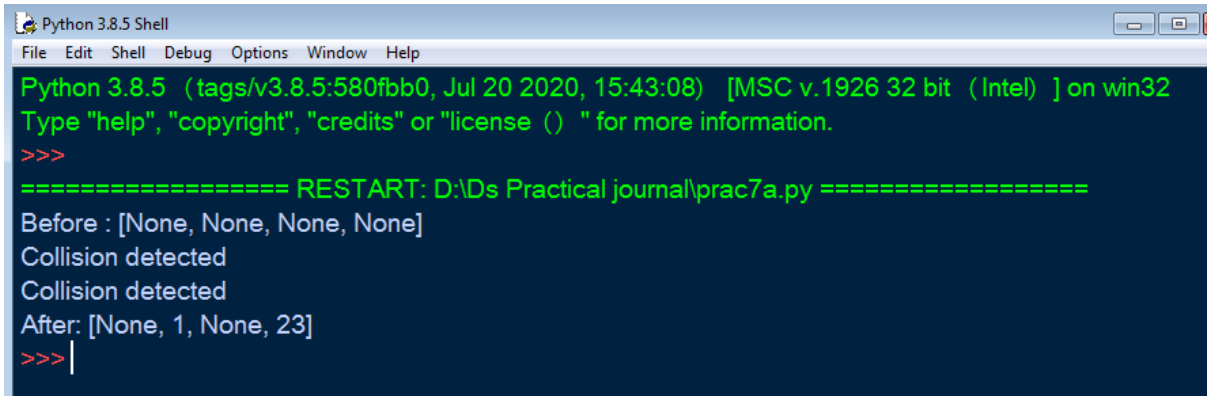
Separate Chaining:

The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

Open Addressing:

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, the size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).





Practical 7b

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

Theory: Linear probing is a scheme in computer programming for resolving collisions in hash tables, data structures for maintaining a collection of key–value pairs and looking up the value associated with a given key. ... Along with quadratic probing and double hashing,linear probing is a form of open addressing

prac7b.py - D:\Ds Practical journal\prac7b.py (3.8.5)

File Edit Format Run Options Window Help

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

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

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


if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [23, 43, 1, 87]
    list_of_list_index = [None]*4
    print ("Before : " + str (list_of_list_index) )
    for value in list_of_keys:
        list_index = Hash (value, 0, len (list_of_keys) ) .get_key_value ()
        print ("Hash value for " + str (value) + " is :" + str (list_index) )
        if list_of_list_index[list_index]:
            print ("Collision detected for " + str (value) )
            if linear_probing:
                old_list_index = list_index
                if list_index == len (list_of_list_index) - 1:
                    list_index = 0
                else:
                    list_index += 1
                list_full = False
                while list_of_list_index[list_index]:
                    if list_index == old_list_index:
                        list_full = True
                        break
                    if list_index + 1 == len (list_of_list_index) :
                        list_index = 0
```

```
            else:
                list__index += 1
        if list__full:
            print ("List was full . Could not save")
        else:
            list__of__list__index[list__index] = value
    else:
        list__of__list__index[list__index] = value
print ("After: " + str (list__of__list__index) )
```

```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)]
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:\Ds Practical journal\prac7b.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]
>>>
```

Practical 8

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

Theory:Inorder:

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

Preorder:

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

Postorder:

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

prac8.py - D:\Ds Practical journal\prac8.py (3.8.5)

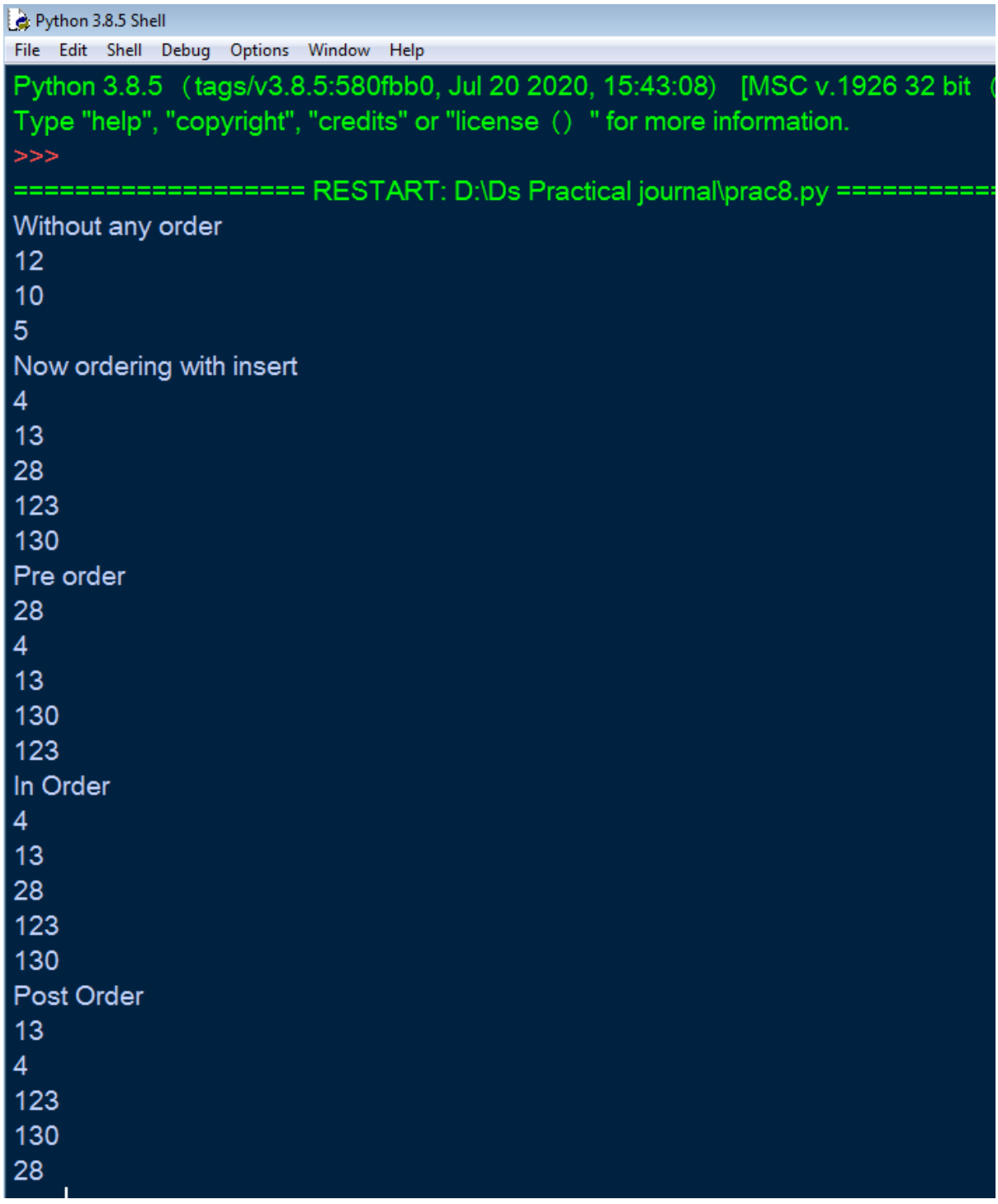
File Edit Format Run Options Window Help

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

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

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

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

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

prac8.py - D:\Ds Practical journal\prac8.py (3.8.5)

File Edit Format Run Options Window Help

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


if___name___ == '___main___ ':
    root = Node (10)
    root.left = Node (12)
    root.right = Node (5)
    print ("Without any order")
    root.PrintTree ()
    root__1 = Node (None)
    root__1.insert (28)
    root__1.insert (4)
    root__1.insert (13)
    root__1.insert (130)
    root__1.insert (123)
    print ("Now ordering with insert")
    root__1.PrintTree ()
    print ("Pre order")
    root__1.Printpreorder ()
    print ("In Order")
    root__1.Printinorder ()
    print ("Post Order")
    root__1.Printpostorder ()
```


Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit
Type "help", "copyright", "credits" or "license() " for more information.
>>>
========== RESTART: D:\Ds Practical journal\prac8.py ==========
Without any order
12
10
5
Now ordering with insert
4
13
28
123
130
Pre order
28
4
13
130
123
In Order
4
13
28
123
130
Post Order
13
4
123
130
28