



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

CERTIFICATE

Name: Mr.Pritom Kalidas Khamaru

Roll No: 373

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)

Subject: Data Structures**INDEX**

Sr No	Date	Topic	Sign
1	04/09/2020	<p>Implement the following for Array:</p> <ul style="list-style-type: none"> 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	<p>Implement the following for Stack:</p> <ul style="list-style-type: none"> 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	<p>Implement the following for Hashing:</p> <ul style="list-style-type: none"> 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.	
---	------------	--	--

Git repository link:- [Data-Structures-practicals](#)

1 Implement the following for Array:

a:

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

Theory

What is searching?

Searching is the process of finding a given value position in a list of values. It decides whether a search key is present in the data or not. It is the algorithmic process of finding a particular item in a collection of items.

What are some searching algorithms?

Linear Search :

A simple approach is to do a linear search, i.e

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
- If x matches with an element, return the index.
- If x doesn't match with any of elements, return -1.

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.

Sorting:

Sorting means arranging the elements of a list in a specific order.

There are many sorting algorithms like:

- Bubble sort
- Merge sort
- Selection Sort
- Insertion Sort
- Quick sort etc

Merging:

Merging means to join two lists.

Reversing:

Reversing means to arrange the elements of the list in reverse order.

In the code below one of the searching and sorting techniques have been applied.

The screenshot shows a terminal window with two panes. The left pane displays a Python script named `practical_1a.py` containing code for array modification. The right pane shows the output of running the script, which includes a traceback, file information, and the resulting sorted list.

```

24 class ArrayModification:
25     def linear_search(self, lst):
26         for i in range(len(lst)):
27             if lst[i] == n:
28                 return f'Position :{i}'
29         return -1
30
31     def insertion_sort(self, lst):
32         for i in range(len(lst)):
33             index = lst[i]
34             k = i - 1
35             while k >= 0 and lst[k] > index:
36                 lst[k + 1] = lst[k]
37                 k -= 1
38             lst[k + 1] = index
39         return lst
40
41     def merge(self, lst1, lst2):
42         return lst1 + lst2
43
44     def reverse(self, lst):
45         return lst[::-1]
46
47 lst = [3, 1, 7, 3, 5, 2]
48 Arrmod = ArrayModification()
49 print(Arrmod.linear_search(lst))
50 print(Arrmod.insertion_sort(lst))
51 print(Arrmod.merge(lst, [12, 13, 14]))
52 print(Arrmod.reverse(lst))

```

```

(base) pritam@pritamPC:~/gitRepos/Data-Structures-practical$ cd gitRepos/Data-Structures-practical
bash: cd: gitRepos/Data-Structures-practical: No such file or directory
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practical$ python practical_1a.py
Position 14
[1, 2, 2, 3, 3, 7, 9]
Traceback (most recent call last):
File "practical_1a.py", line 34, in <module>
    print(Arrmod.merge(lst,[12,13,14]))
File "practical_1a.py", line 25, in merge
    return ArrayModification.insertion_sort(lst1 + lst2)
TypeError: insertion_sort() missing 1 required positional argument: 'lst'
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practical$ python practical_1a.py
[1, 2, 2, 3, 3, 7, 9]
[1, 2, 2, 3, 3, 7, 9, 12, 13, 14]
[9, 7, 5, 3, 2, 2, 1]
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practical$ 

```

B:

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

Theory

Matrix Addition: Matrix addition is the operation of adding two matrices by adding the corresponding entries together. The matrix can be added only when the number of rows and columns of the first matrix is equal to the number of rows and columns of the second matrix.

Matrix Multiplication: We can multiply two matrices if, and only if, the number of columns in the first matrix equals the number of rows in the second matrix. Otherwise, the product of two matrices is undefined.

Matrix Transpose: If $A=[a_{ij}]$ be a matrix of order $m \times n$, then the matrix obtained by interchanging the rows and columns of A is known as Transpose of matrix A . Transpose of matrix A is represented by A^T .

Code and Output:

The screenshot shows a Sublime Text window with multiple tabs open. The current tab is 'practical_1b.py' which contains Python code for matrix operations. The code includes definitions for Mat1, Mat2, and Mat3, and functions for Matrix Addition, Matrix Multiplication, and matrix transpose. The output of the code execution is displayed in the bottom panel, showing the resulting matrices and the completion message '[Finished in 0.0s]'. The left sidebar shows a file tree for a 'Data-Structures-practicals' directory.

```
Oct 25 13:47 • -/gitRepos/Data-Structures-practicals/practical_1b.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)
```

```
1 Mat1 = [[3, 4, -6],  
2      [12, 71, 24],  
3      [21, 3, 21]]  
4  
5 Mat2 = [[2, 16, -16],  
6      [1, 7, 3],  
7      [-1, 3, 3]]  
8 Mat3 = [[0, 0, 0],  
9      [0, 0, 0],  
10     [0, 0, 0]]  
11  
12 # Matrix Addition  
13 for i in range(len(Mat1)):  
14     for j in range(len(Mat2[0])):  
15         for k in range(len(Mat2)):.  
16             Mat3[i][j] += Mat1[i][k] + Mat2[k][j]  
17  
18 print(Mat3)  
19  
20 # Matrix Multiplication  
21 Mat3 = [[0, 0, 0, 0],  
22      [0, 0, 0, 0],  
23      [0, 0, 0, 0]]  
24  
25 for i in range(len(Mat1)):  
26     for j in range(len(Mat2[0])):  
27         for k in range(len(Mat2)):  
28             Mat3[i][j] += Mat1[i][k] * Mat2[k][j]  
29  
30 print(Mat3)  
31  
32 #matrix transpose  
33 for i in map(list, zip(*Mat1)):  
34     print(i)  
35  
36
```

```
[[3, 27, -15], [109, 133, 91], [47, 71, 29]]  
[[16, 58, -78, 0], [71, 761, -333, 0], [24, 420, -282, 0]]  
[3, 12, 21]  
[4, 71, 3]  
[-6, 24, 21]  
[Finished in 0.0s]
```

2.

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

Theory

Singly Linked List:

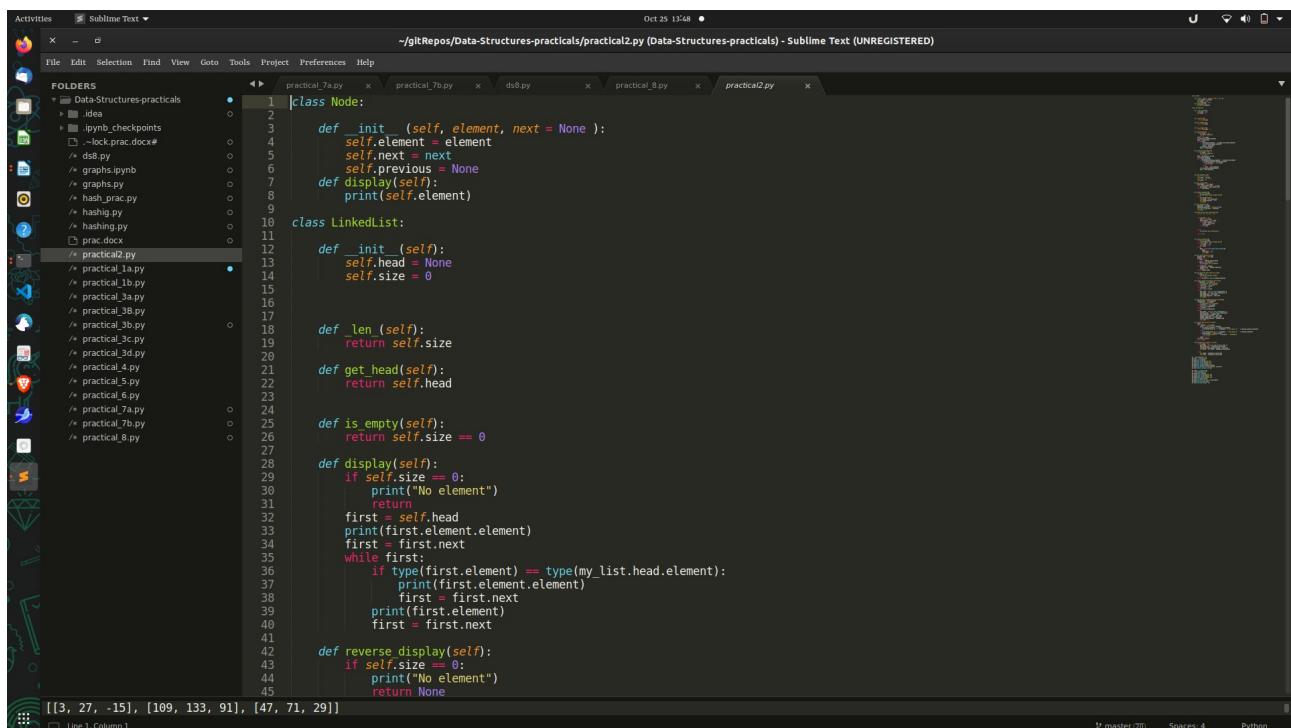
A singly linked list, in its simplest form, is a collection of nodes that collectively form a linear sequence. Each node stores a reference to an object that is an element of the sequence, as well as a reference to the next node of the list

Doubly Linked List:

In a singly linked list, each node maintains a reference to the node that is immediately after it. However, there are limitations that stem from the asymmetry of a singly linked list. To provide greater symmetry, we define a linked list in which each node keeps an explicit reference to the node before it and a reference to the node after it.

Such a structure is known as a doubly linked list. These lists allow a greater variety of $O(1)$ -time update operations, including insertions and deletions at arbitrary positions within the list. We continue to use the term “next” for the reference to the node that follows another, and we introduce the term “prey” for the reference to the node that precedes it. With array-based sequences, an integer index was a convenient means for describing a position within a sequence. However, an index is not convenient for linked lists as there is no efficient way to find the j th element; it would seem to require a traversal of a portion of the list.

When working with a linked list, the most direct way to describe the location of an operation is by identifying a relevant node of the list. However, we prefer to encapsulate the inner workings of our data structure to avoid having users directly access nodes of a list.



```

Activities Sublime Text ▾
File Edit Selection Find View Goto Tools Project Preferences Help
Oct 25 13:48 • -/gitRepos/Data-Structures-practicals/practical2.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)
FOLDERS
  Data-Structures-practicals
    idea
      .ipynb_checkpoints
        ~lock_prac.ipynb
      dsb.py
      graphs.ipynb
      graphs.py
      hash_prac.py
      haship.py
      hashing.py
      prac.docx
      practical2.py
    practical_1a.py
    practical_1b.py
    practical_3a.py
    practical_3B.py
    practical_3b.py
    practical_3c.py
    practical_3d.py
    practical_4.py
    practical_5.py
    practical_6.py
    practical_7a.py
    practical_7b.py
    practical_8.py
    practical2.py
practical2.py
  1  class Node:
  2
  3      def __init__(self, element, next = None):
  4          self.element = element
  5          self.next = next
  6          self.previous = None
  7      def display(self):
  8          print(self.element)
  9
 10  class LinkedList:
 11
 12      def __init__(self):
 13          self.head = None
 14          self.size = 0
 15
 16      def __len__(self):
 17          return self.size
 18
 19      def get_head(self):
 20          return self.head
 21
 22      def is_empty(self):
 23          return self.size == 0
 24
 25      def display(self):
 26          if self.size == 0:
 27              print("No element")
 28          return
 29          first = self.head
 30          print(first.element.element)
 31          first = first.next
 32          while first:
 33              if type(first.element) == type(my_list.head.element):
 34                  print(first.element.element)
 35              first = first.next
 36          print(first.element)
 37          first = first.next
 38
 39      def reverse_display(self):
 40          if self.size == 0:
 41              print("No element")
 42          return None
 43
 44
 45
[[3, 27, -15], [109, 133, 91], [47, 71, 29]]
Line 1, Column 1
master (25) Spaces: 4 Python

```

Activities Sublime Text • Oct 25 13:48 • -/gitRepos/Data-Structures-practicals/practical2.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
Data-Structures-practicals
  - idea
  - ipynb_checkpoints
    - lock_prac.docx#
    - dsb.py
    - graphs.ipynb
    - graphs.py
    - hash_prac.py
    - haship.py
    - hashing.py
    - prac.docx
    practical2.py
    practical_1a.py
    practical_1b.py
    practical_3a.py
    practical_3B.py
    practical_3b.py
    practical_3c.py
    practical_3d.py
    practical_4.py
    practical_5.py
    practical_6.py
    practical_7a.py
    practical_7b.py
    practical_8.py
practical2.py

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)
        if last.previous == self.head:
            return None
        else:
            last = last.previous
    print(last.previous)
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)

```

Activities Sublime Text • Oct 25 13:49 • -/gitRepos/Data-Structures-practicals/practical2.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
Data-Structures-practicals
  - idea
  - ipynb_checkpoints
    - lock_prac.docx#
    - dsb.py
    - graphs.ipynb
    - graphs.py
    - hash_prac.py
    - haship.py
    - hashing.py
    - prac.docx
    practical2.py
    practical_1a.py
    practical_1b.py
    practical_3a.py
    practical_3B.py
    practical_3b.py
    practical_3c.py
    practical_3d.py
    practical_4.py
    practical_5.py
    practical_6.py
    practical_7a.py
    practical_7b.py
    practical_8.py
practical2.py

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")

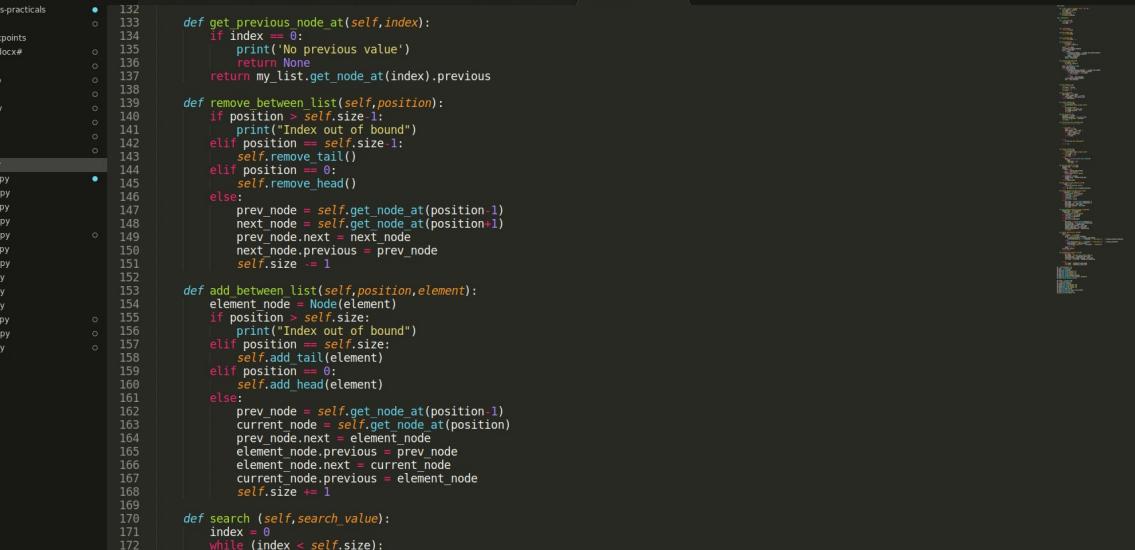
    return None

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:
        print("Index out of bound")
        return None
    while(counter < index):
        element_node = element_node.next
        counter += 1
    return element_node

```

[[3, 27, -15], [109, 133, 91], [47, 71, 29]]



The screenshot shows a Sublime Text window with the following details:

- File Path:** ~/gitRepos/Data-Structures-practicals/practical2.py
- File Type:** Data-Structures-practicals (UNREGISTERED)
- Code Content:** A Python script for a linked list implementation. The code includes methods for getting the previous node, removing nodes between two positions, adding elements between two positions, and searching for specific values.
- Code Snippet:**

```
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 type(value.element) == type(my_list.head):
            print("Searching at " + str(index) + " and value is " + str(value.element.element))
        else:
```
- Bottom Status Bar:** Line 1 Column 1, master (20), Spaces: 4, Python

The screenshot shows a Sublime Text window with multiple tabs open, displaying a Python script named `practical2.py`. The script contains functions for searching and merging linked lists. The code uses print statements to output the state of the lists during execution.

```
Oct 25 13:51 • -/gitRepos/Data-Structures-practicals/practical2.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)
```

```
print("Searching at " + str(index) + " and value is " + str(value.element))
if value.element == search_value:
    print("Found value at " + str(index) + " location")
    return True
index += 1
print("Not Found")
return False

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

ll = Node('Element 1')
my_list = LinkedList()
my_list.add_head(ll)
my_list.add_tail('Element 2')
my_list.add_tail('Element 3')
my_list.add_tail('Element 4')
my_list.get_head().element.element
my_list.add_between(list(2), 'Element between')
my_list.remove_between(list(2))

my_list2 = LinkedList()
l2 = Node('Element 5')
my_list2.add_head(l2)
my_list2.add_tail('Element 6')
my_list2.add_tail('Element 7')
my_list2.add_tail('Element 8')
my_list2.add_tail('Element 9')
my_list.merge(my_list2)
my_list.get_previous_node_at(3).element
my_list.reverse_display()
my_list.search('Element 6')
```

```
2023-01-10 14:23:00.000 my_list.add(4); Elements: 4
Element 8
Element 7
Element 6
Element 5
Element 5
Element 3
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
[Finished in 0.0s]
```

3. Implement the following for Stack:

a.

Aim: Perform Stack operations using Array implementation.

Theory

Stack:

A stack is a collection of objects that are inserted and removed according to the last-in, first-out (LIFO) principle. A user may insert objects into a stack at any time, but may only access or remove the most recently inserted object that remains (at the so-called “top” of the stack). We can implement a stack quite easily by storing its elements in a Python list. The list class already supports adding an element to the end with the append method, and removing the last element with the pop method, so it is natural to align the top of the stack at the end of the list.

Stack is an abstract data type (ADT) such that an instance S supports the following two methods:

S.push(e): Add element e to the top of stack S.

S.pop(): Remove and return the top element from the stack S;

an error occurs if the stack is empty.

Code and output:

The screenshot shows a Sublime Text editor window with multiple tabs open. The current tab contains Python code for a Stack class. The code defines a class Stack with methods for push, pop, get_head, and display. It uses a list called stack_arr to store elements. The code is as follows:

```

1  class Stack:
2      def __init__(self):
3          self.stack_arr = []
4
5      def push(self, value):
6          self.stack_arr.append(value)
7
8      def pop(self):
9          if len(self.stack_arr) == 0:
10              print('Stack is empty!')
11              return None
12          else:
13              self.stack_arr.pop()
14
15      def get_head(self):
16          if len(self.stack_arr) == 0:
17              print('Stack is empty!')
18              return None
19          else:
20              return self.stack_arr[-1]
21
22      def display(self):
23          if len(self.stack_arr) == 0:
24              print('Stack is empty!')
25              return None
26          else:
27              print(self.stack_arr)
28
29
30 stack = Stack()
31 stack.push(1)
32 stack.push(3)
33 stack.push(5)
34 stack.pop()
35 stack.display()
36 stack.get_head()
37

```

The status bar at the bottom indicates the code has finished execution in 0.0s. The output pane shows the result: [1, 3].

b.

Aim: Implement Tower of Hanoi

Theory:

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

1) Only one disk can be moved at a time.

2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

3) No disk may be placed on top of a smaller disk.

To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser number of disks, say → 1 or 2. We mark three towers with name, source, destination and aux (only to help moving the disks). If we have only one disk, then it can easily be moved from source to destination peg.

If we have 2 disks –

First, we move the smaller (top) disk to aux peg.

Then, we move the larger (bottom) disk to destination peg.

And finally, we move the smaller disk from aux to destination peg.

So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks. We divide the stack of disks in two parts. The largest disk (nth disk) is in one part and all other (n-1) disks are in the second part.

Our ultimate aim is to move disk n from source to destination and then put all other (n1) disks onto it. We can imagine to apply the same in a recursive way for all given set of disks. Each peg is a Stack object.

Code and output

```

#> tower of hanoi
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:
            return 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 towerOfHanoi(n, fromrod,to,temp):
    if n == 1:
        fromrod.pop()
        to.push('disk 1')
        if to.display() != None:
            print(to.display())
    else:
        towerOfHanoi(n-1, fromrod, temp, to)
        fromrod.pop()

```

c.

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

Theory

Different operations can be performed on the polynomials like addition, subtraction, multiplication, and division. A polynomial is an expression within which a finite number of constants and variables are combined using addition, subtraction, multiplication, and exponents. Adding and subtracting polynomials is just adding and subtracting their like terms. The sum of two monomials is called a binomial and the sum of three monomials is called a trinomial. The sum of a finite number of monomials in x is called a polynomial in x . The coefficients of the monomials in a polynomial are called the coefficients of the polynomial. If all the coefficients of a polynomial are zero, then the polynomial is called the zero polynomial.

Two polynomials can be added by using arithmetic operator plus (+). Adding polynomials is simply “combining like terms” and then add the like terms.

Every Polynomial in the program is a Doubly Linked List object. The corresponding terms are added and displayed in the form of an expression.

```

class Node:
    def __init__(self, element, next = None):
        self.element = element
        self.next = next
        self.previous = None
    def display(self):
        print(self.element)

class LinkedList:
    def __init__(self):
        self.head = None
        self.size = 0

    def len_(self):
        return self.size

    def get_head(self):
        return self.head

    def is_empty(self):
        return self.size == 0

    def display(self):
        if self.size == 0:
            print("No element")
            return
        first = self.head
        print(first.element.element)
        first = first.next
        while first:
            if type(first.element) == type(my.list.head.element):
                print(first.element.element)
            first = first.next
            print(first.element)
        first = first.next

    def reverse_display(self):
        if self.size == 0:
            print("No element")
            return None

```

```

B = Stack()
C = Stack()
def towerOfHanoi(n, fromrod, to, temp):
    if n == 1:
        fromrod.pop()
        to.push('disk 1')
        if to.display() != None:
            print(to.display())
    else:
        towerOfHanoi(n-1, fromrod, temp, to)
        fromrod.pop()
        to.push('disk ' + str(n))
        if to.display() != None:
            print(to.display())
        towerOfHanoi(n-1, temp, to, fromrod)
n = 3
for i in range(n):
    A.push('disk ' + str(i+1))
towerOfHanoi(n, A, C, B)

```

```

['disk 1']
['disk 2']
['disk 2', 'disk 1']
['disk 3']
['disk 1']
['disk 3', 'disk 2']
['disk 3', 'disk 2', 'disk 1']
[Finished in 0.0s]

```

Activities Sublime Text • Oct 25 16:05 • -/gitRepos/Data-Structures-practicals/practical_3c.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS practical_7a.py x practical_7b.py x ds8.py x practical_8.py x practical_3B.py x practical_3c.py x
Data-Structures-practicals
  .idea
  ipynb_checkpoints
  ~lock_prac.docx#
  dsb.py
  graphs.ipynb
  graphs.py
  hash_prac.py
  hash.py
  hashing.py
  prac.docx
  practical2.py
  practical1a.py
  practical1b.py
  practical3a.py
  practical3B.py
  practical3b.py
  practical3c.py
  practical3d.py
  practical4.py
  practical5.py
  practical6.py
  practical7a.py
  practical7b.py
  practical8.py
practical_3c.py
  41
  def reverse_display(self):
  42      if self.size == 0:
  43          print("No element")
  44          return None
  45      last = my_list.get_tail()
  46      print(last.element)
  47      while last.previous:
  48          if type(last.previous.element) == type(my_list.head):
  49              print((last.previous.element))
  50              if last.previous == self.head:
  51                  last = None
  52              else:
  53                  last = last.previous
  54      print(last.previous.element)
  55      last = last.previous
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000
  1001
  1002
  1003
  1004
  1005
  1006
  1007
  1008
  1009
  1010
  1011
  1012
  1013
  1014
  1015
  1016
  1017
  1018
  1019
  1020
  1021
  1022
  1023
  1024
  1025
  1026
  1027
  1028
  1029
  1030
  1031
  1032
  1033
  1034
  1035
  1036
  1037
  1038
  1039
  1040
  1041
  1042
  1043
  1044
  1045
  1046
  1047
  1048
  1049
  1050
  1051
  1052
  1053
  1054
  1055
  1056
  1057
  1058
  1059
  1060
  1061
  1062
  1063
  1064
  1065
  1066
  1067
  1068
  1069
  1070
  1071
  1072
  1073
  1074
  1075
  1076
  1077
  1078
  1079
  1080
  1081
  1082
  1083
  1084
  1085
  1086
  1087
  1088
  1089
  1090
  1091
  1092
  1093
  1094
  1095
  1096
  1097
  1098
  1099
  1100
  1101
  1102
  1103
  1104
  1105
  1106
  1107
  1108
  1109
  1110
  1111
  1112
  1113
  1114
  1115
  1116
  1117
  1118
  1119
  1120
  1121
  1122
  1123
  1124
  1125
  1126
  1127
  1128
  1129
  1130
  1131
  1132
  1133
  1134
  1135
  1136
  1137
  1138
  1139
  1140
  1141
  1142
  1143
  1144
  1145
  1146
  1147
  1148
  1149
  1150
  1151
  1152
  1153
  1154
  1155
  1156
  1157
  1158
  1159
  1160
  1161
  1162
  1163
  1164
  1165
  1166
  1167
  1168
  1169
  1170
  1171
  1172
  1173
  1174
  1175
  1176
  1177
  1178
  1179
  1180
  1181
  1182
  1183
  1184
  1185
  1186
  1187
  1188
  1189
  1190
  1191
  1192
  1193
  1194
  1195
  1196
  1197
  1198
  1199
  1200
  1201
  1202
  1203
  1204
  1205
  1206
  1207
  1208
  1209
  1210
  1211
  1212
  1213
  1214
  1215
  1216
  1217
  1218
  1219
  1220
  1221
  1222
  1223
  1224
  1225
  1226
  1227
  1228
  1229
  1230
  1231
  1232
  1233
  1234
  1235
  1236
  1237
  1238
  1239
  1240
  1241
  1242
  1243
  1244
  1245
  1246
  1247
  1248
  1249
  1250
  1251
  1252
  1253
  1254
  1255
  1256
  1257
  1258
  1259
  1260
  1261
  1262
  1263
  1264
  1265
  1266
  1267
  1268
  1269
  1270
  1271
  1272
  1273
  1274
  1275
  1276
  1277
  1278
  1279
  1280
  1281
  1282
  1283
  1284
  1285
  1286
  1287
  1288
  1289
  1290
  1291
  1292
  1293
  1294
  1295
  1296
  1297
  1298
  1299
  1300
  1301
  1302
  1303
  1304
  1305
  1306
  1307
  1308
  1309
  1310
  1311
  1312
  1313
  1314
  1315
  1316
  1317
  1318
  1319
  1320
  1321
  1322
  1323
  1324
  1325
  1326
  1327
  1328
  1329
  1330
  1331
  1332
  1333
  1334
  1335
  1336
  1337
  1338
  1339
  1340
  1341
  1342
  1343
  1344
  1345
  1346
  1347
  1348
  1349
  1350
  1351
  1352
  1353
  1354
  1355
  1356
  1357
  1358
  1359
  1360
  1361
  1362
  1363
  1364
  1365
  1366
  1367
  1368
  1369
  1370
  1371
  1372
  1373
  1374
  1375
  1376
  1377
  1378
  1379
  1380
  1381
  1382
  1383
  1384
  1385
  1386
  1387
  1388
  1389
  1390
  1391
  1392
  1393
  1394
  1395
  1396
  1397
  1398
  1399
  1400
  1401
  1402
  1403
  1404
  1405
  1406
  1407
  1408
  1409
  1410
  1411
  1412
  1413
  1414
  1415
  1416
  1417
  1418
  1419
  1420
  1421
  1422
  1423
  1424
  1425
  1426
  1427
  1428
  1429
  1430
  1431
  1432
  1433
  1434
  1435
  1436
  1437
  1438
  1439
  1440
  1441
  1442
  1443
  1444
  1445
  1446
  1447
  1448
  1449
  1450
  1451
  1452
  1453
  1454
  1455
  1456
  1457
  1458
  1459
  1460
  1461
  1462
  1463
  1464
  1465
  1466
  1467
  1468
  1469
  1470
  1471
  1472
  1473
  1474
  1475
  1476
  1477
  1478
  1479
  1480
  1481
  1482
  1483
  1484
  1485
  1486
  1487
  1488
  1489
  1490
  1491
  1492
  1493
  1494
  1495
  1496
  1497
  1498
  1499
  1500
  1501
  1502
  1503
  1504
  1505
  1506
  1507
  1508
  1509
  1510
  1511
  1512
  1513
  1514
  1515
  1516
  1517
  1518
  1519
  1520
  1521
  1522
  1523
  1524
  1525
  1526
  1527
  1528
  1529
  1530
  1531
  1532
  1533
  1534
  1535
  1536
  1537
  1538
  1539
  1540
  1541
  1542
  1543
  1544
  1545
  1546
  1547
  1548
  1549
  1550
  1551
  1552
  1553
  1554
  1555
  1556
  1557
  1558
  1559
  1560
  1561
  1562
  1563
  1564
  1565
  1566
  1567
  1568
  1569
  1570
  1571
  1572
  1573
  1574
  1575
  1576
  1577
  1578
  1579
  1580
  1581
  1582
  1583
  1584
  1585
  1586
  1587
  1588
  1589
  1590
  1591
  1592
  1593
  1594
  1595
  1596
  1597
  1598
  1599
  1600
  1601
  1602
  1603
  1604
  1605
  1606
  1607
  1608
  1609
  1610
  1611
  1612
  1613
  1614
  1615
  1616
  1617
  1618
  1619
  1620
  1621
  1622
  1623
  1624
  1625
  1626
  1627
  1628
  1629
  1630
  1631
  1632
  1633
  1634
  1635
  1636
  1637
  1638
  1639
  1640
  1641
  1642
  1643
  1644
  1645
  1646
  1647
  1648
  1649
  1650
  1651
  1652
  1653
  1654
  1655
  1656
  1657
  1658
  1659
  1660
  1661
  1662
  1663
  1664
  1665
  1666
  1667
  1668
  1669
  1670
  1671
  1672
  1673
  1674
  1675
  1676
  1677
  1678
  1679
  1680
  1681
  1682
  1683
  1684
  1685
  1686
  1687
  1688
  1689
  1690
  1691
  1692
  1693
  1694
  1695
  1696
  1697
  1698
  1699
  1700
  1701
  1702
  1703
  1704
  1705
  1706
  1707
  1708
  1709
  1710
  1711
  1712
  1713
  1714
  1715
  1716
  1717

```

Activities Sublime Text • Oct 25 16:06 • /gitRepos/Data-Structures-practicals/practical_3c.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

- └ Data-Structures-practicals
 - └ idea
 - └ ipynb_checkpoints
 - └ ~lock_prac.docx#
 - └ dsb.py
 - └ graphs.ipynb
 - └ hash_prac.py
 - └ hashipg.py
 - └ hashing.py
 - └ prac.docx
 - └ practical2.py
 - └ practical1a.py
 - └ practical1b.py
 - └ practical3a.py
 - └ practical3b.py
 - └ practical3c.py
 - └ practical3d.py
 - └ practical4.py
 - └ practical5.py
 - └ practical6.py
 - └ practical7a.py
 - └ practical7b.py
 - └ practical7c.py
 - └ practical7d.py
 - └ practical8.py

```

128     return None
129     while(counter < index):
130         element_node = element_node.next
131         counter += 1
132     return element_node
133
134     def get previous_node_at(self, index):
135         if index == 0:
136             print("No previous value")
137             return None
138         return my_list.get_node_at(index).previous
139
140     def remove_between list(self, position):
141         if position > self.size-1:
142             print("Index out of bound")
143         elif position == self.size-1:
144             self.remove_tall()
145         elif position == 0:
146             self.remove_head()
147         else:
148             prev_node = self.get_node_at(position-1)
149             next_node = self.get_node_at(position+1)
150             prev_node.next = next_node
151             next_node.previous = prev_node
152             self.size -= 1
153
154     def add_between list(self, position, element):
155         element_node = Node(element)
156         if position > self.size:
157             print("Index out of bound")
158         elif position == self.size:
159             self.add_tall(element)
160         elif position == 0:
161             self.add_head(element)
162         else:
163             prev_node = self.get_node_at(position-1)
164             current_node = self.get_node_at(position)
165             prev_node.next = element_node
166             element_node.previous = prev_node
167             element_node.next = current_node
168             current_node.previous = element_node
169             self.size += 1
170
171     def search (self, search_value):
172         index = 0

```

[disk 1] Line 29, Column 27 Oct 25 16:06 master Spaces: 4 Python

Activities Sublime Text • Oct 25 16:06 • /gitRepos/Data-Structures-practicals/practical_3c.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

- └ Data-Structures-practicals
 - └ idea
 - └ ipynb_checkpoints
 - └ ~lock_prac.docx#
 - └ dsb.py
 - └ graphs.ipynb
 - └ hash_prac.py
 - └ hashipg.py
 - └ hashing.py
 - └ prac.docx
 - └ practical2.py
 - └ practical1a.py
 - └ practical1b.py
 - └ practical3a.py
 - └ practical3b.py
 - └ practical3c.py
 - └ practical3d.py
 - └ practical4.py
 - └ practical5.py
 - └ practical6.py
 - └ practical7a.py
 - └ practical7b.py
 - └ practical7c.py
 - └ practical7d.py
 - └ practical8.py

```

167     current_node.previous = element_node
168     self.size += 1
169
170     def search (self, search_value):
171         index = 0
172         while (index < self.size):
173             value = self.get_node_at(index)
174             if value.element == search_value:
175                 return value.element
176             index += 1
177         print("Not Found")
178         return False
179
180     def merge(self, linkedlist_value):
181         if self.size == 0:
182             last_node = self.get_node_at(self.size-1)
183             last_node.next = linkedlist_value.head
184             linkedlist_value.head.previous = last_node
185             self.size = self.size + linkedlist_value.size
186
187         else:
188             self.head = linkedlist_value.head
189             self.size = linkedlist_value.size
190
191         my_list = LinkedList()
192         order = int(input("Enter the order for polynomial : "))
193         my_list.add_head(Node(int(input("Enter coefficient for power {order} : "))))
194         for i in reversed(range(order)):
195             my_list.add_tail(int(input("Enter coefficient for power {i} : ")))
196
197         my_list2 = LinkedList()
198         my_list2.add_head(Node(int(input("Enter coefficient for power {order} : "))))
199         for i in reversed(range(order)):
200             my_list2.add_tail(int(input("Enter coefficient for power {i} : ")))
201
202         for i in range(order + 1):
203             print(my_list.get_node_at(i).element + my_list2.get_node_at(i).element)
204
205
206

```

[disk 1] Line 29, Column 27 Oct 25 16:06 master Spaces: 4 Python

```
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$ python practical_3c.py
Enter the order for polynomial : 3 188           self.head = linkedlist_value.head
Enter coefficient for power 3 : 2 189           self.size = linkedlist_value.size
Enter coefficient for power 2 : 1 190
Enter coefficient for power 1 : 0 191
Enter coefficient for power 0 : 1 192
Enter coefficient for power 3 : 4 193   my_list = LinkedList()
Enter coefficient for power 2 : 3 194   order = int(input('Enter the order for polynomial : '))
Enter coefficient for power 1 : 2 195   my_list.add_head(Node(int(input("Enter coefficient for power {order} : "))))
Enter coefficient for power 0 : 1 196   for i in reversed(range(order)):
6   /* practical_6.py               197     my_list.add_tail(int(input("Enter coefficient for power {i} : ")))
4   /* practical_7a.py             198
2   /* practical_7b.py             199   my_list2 = LinkedList()
                                         200   for i in reversed(range(order)):
                                         201     my_list2.add_head(int(input("Enter coefficient for power {order} : ")))
                                         202     my_list2.add_tail(int(input("Enter coefficient for power {i} : ")))

(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$
```

d.

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

Theory

Factorial:

The factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is $1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$.

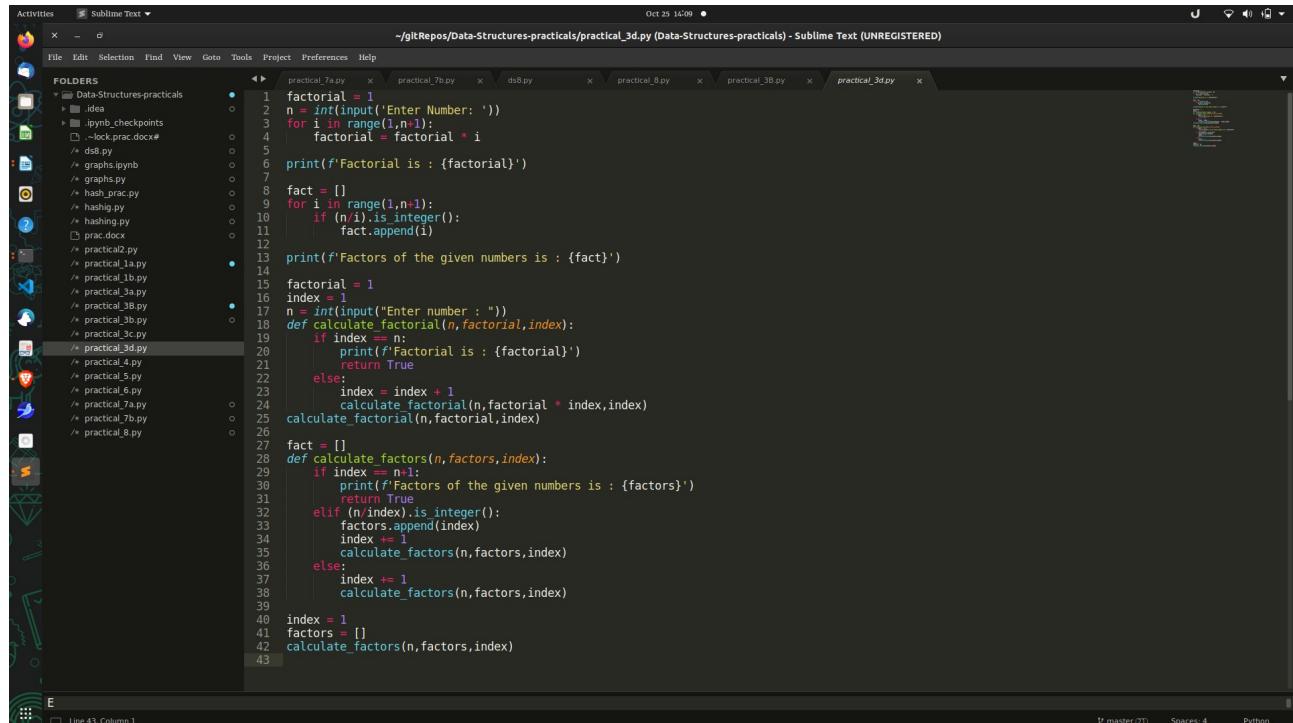
Factorial is not defined for negative numbers and the factorial of zero is one, $0! = 1$.

You can find it using recursion as well as iteration to calculate the factorial of a number.

Factorial:

Factors are the numbers you multiply to get another number. For instance, factors of 15 are 3 and 5, because $3 \times 5 = 15$. Some numbers have more than one factorization (more than one way of being factored). For instance, 12 can be factored as 1×12 , 2×6 , or 3×4 . A number that can only be factored as 1 time itself is called "prime".

You can find it using recursion as well as iteration to calculate the factors of a number.



The screenshot shows a Sublime Text editor window with multiple tabs open. The current tab is 'practical_3d.py'. The code implements two functions: 'calculate_factorial' and 'calculate_factors'. The 'calculate_factorial' function takes an integer 'n' and returns its factorial. The 'calculate_factors' function takes an integer 'n' and returns a list of factors of 'n'. Both functions use recursion and dynamic programming to avoid recalculating intermediate values.

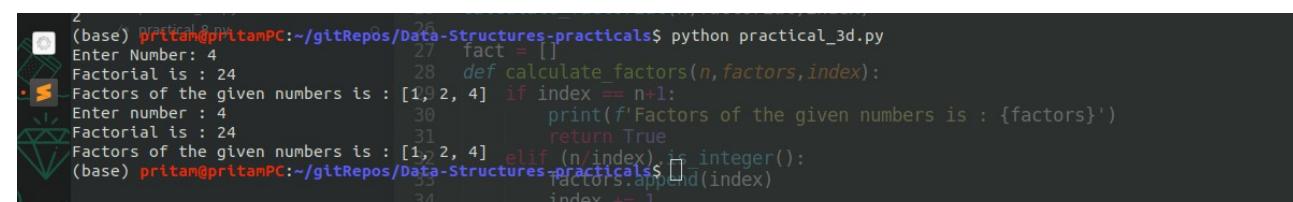
```
factorial = 1
n = int(input('Enter Number: '))
for i in range(1,n+1):
    factorial *= i
print(f'Factorial is : {factorial}')

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)

factors = []
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)
```



The terminal window shows the execution of the script. It prompts for an input number (4), calculates the factorial (24), and then lists the factors of 4 (1, 2, 4).

```
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$ python practical_3d.py
Enter Number: 4
Factorial is : 24
Factors of the given numbers is : [1, 2, 4]
Enter number : 4
Factorial is : 24
Factors of the given numbers is : [1, 2, 4]
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$
```

4.**Aim: Perform Queues operations using Circular Array implementation.****Theory****Queue**

the queue abstract data type defines a collection that keeps objects in a sequence, where element access and deletion are restricted to the first element in the queue, and element insertion is restricted to the back of the sequence. This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in, first-out (FIFO) principle. The queue abstract data type (ADT) supports the following two fundamental methods for a queue Q:

Q.enqueue(e): Add element e to the back of queue Q.

Q.dequeue(): Remove and return the first element from queue Q;
an error occurs if the queue is empty.

For the stack ADT, we created a very simple adapter class that used a Python list as the underlying storage.

Double Ended Queue

We next consider a queue-like data structure that supports insertion and deletion at both the front and the back of the queue. Such a structure is called a double ended queue, or deque, which is usually pronounced “deck” to avoid confusion with the dequeue method of the regular queue ADT, which is pronounced like the abbreviation “D.Q.”

The deque abstract data type is more general than both the stack and the queue ADTs.

```

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

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

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

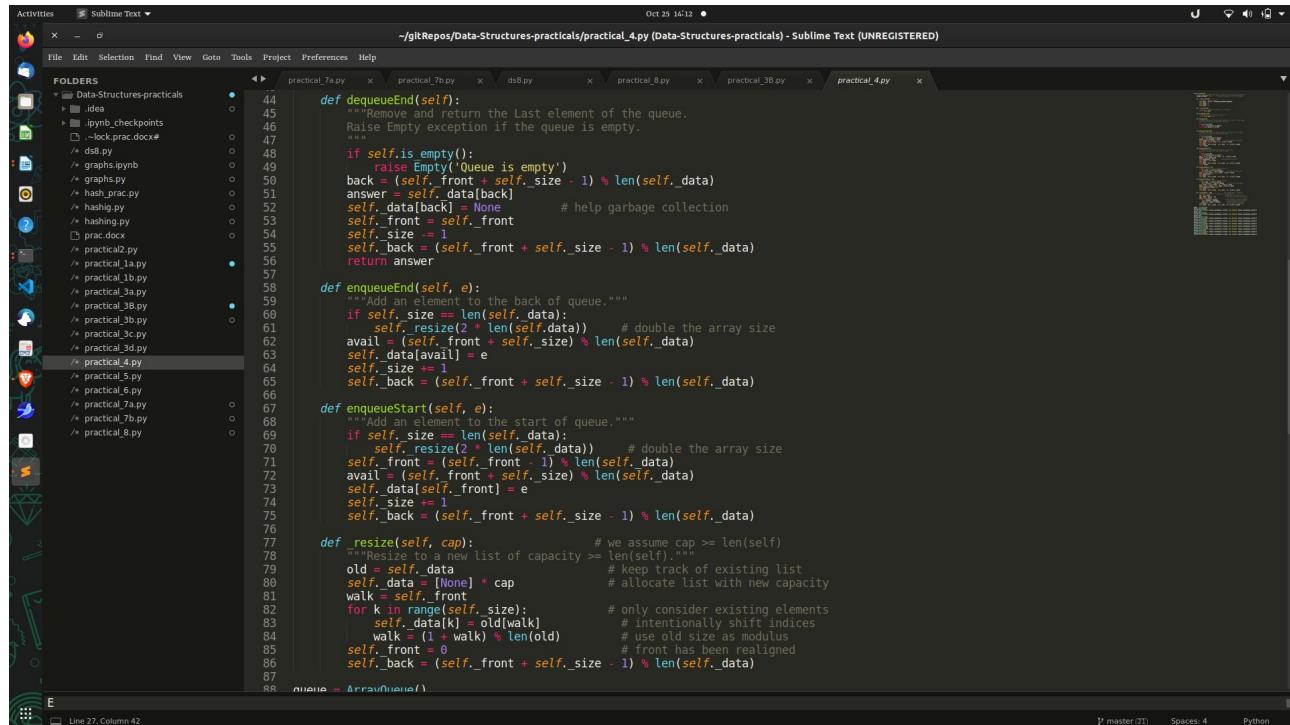
    def is_empty(self):
        """Return True if the queue is empty."""
        return self._size == 0

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

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

    def dequeueEnd(self):
        """Remove and return the last element of the queue.
        """

```



```

Oct 25 16:12 • -/gitRepos/Data-Structures-practicals/practical_4.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
practical_7a.py x practical_7b.py x ds8.py x practical_8.py x practical_3b.py x practical_4.py x

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)

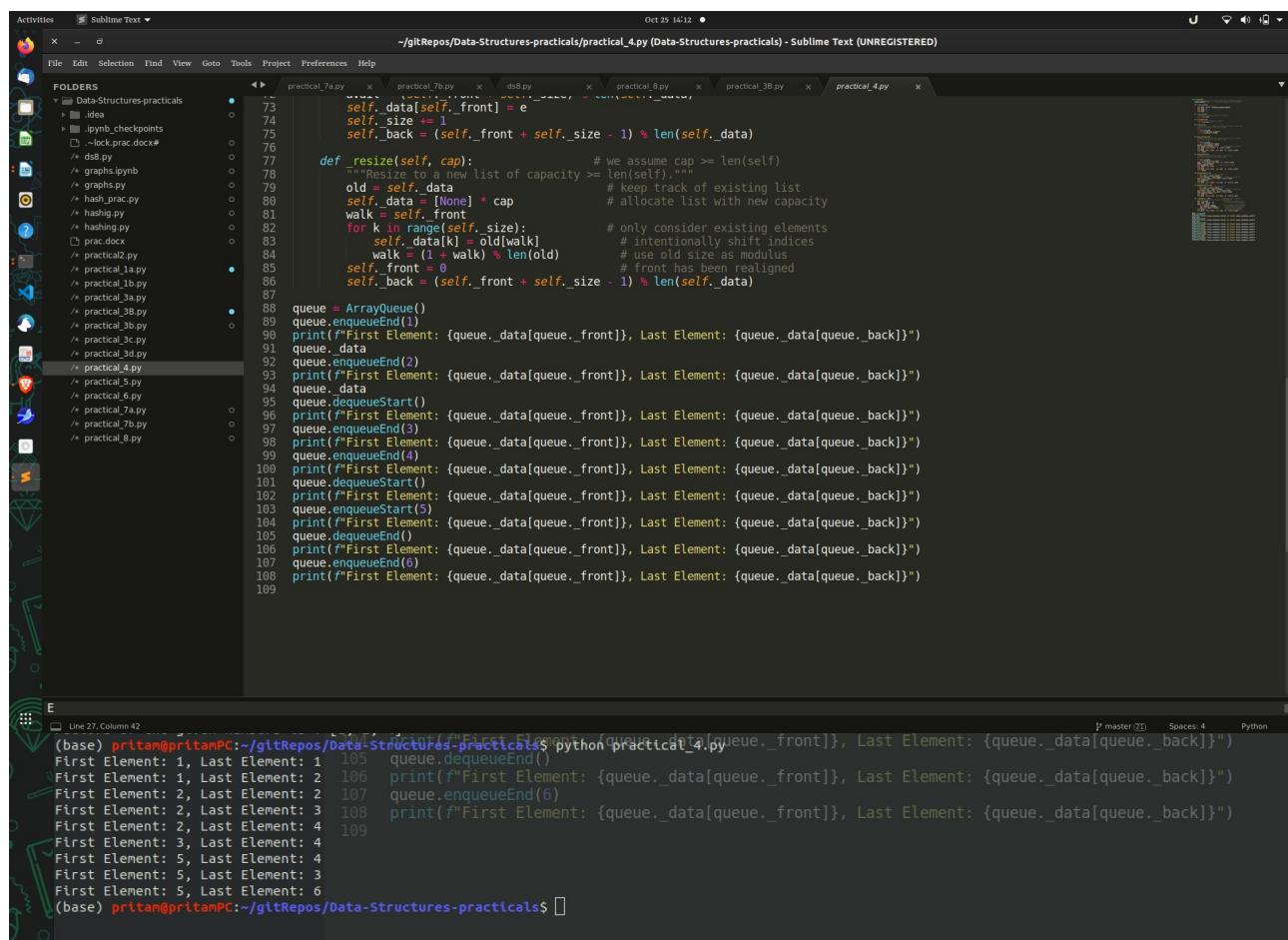
def enqueueStart(self, e):
    """Add an element to the start of queue."""
    if self.size == len(self._data):
        self.resize(2 * len(self._data)) # double the array size
    self.front = (self.front - 1) % len(self._data)
    avail = (self.front + self.size) % len(self._data)
    self._data[self.front] = e
    self.size += 1
    self.back = (self.front + self.size - 1) % len(self._data)

def _resize(self, cap):
    """Resize to a new list of capacity >= len(self)."""
    old = self._data
    self._data = [None] * cap
    self._data[:len(old)] = old
    self.front = 0
    self.back = (self.front + self.size - 1) % len(self._data)
    walk = self.front
    for k in range(self.size):
        self._data[k] = old[walk] # intentionally shift indices
        walk = (1 + walk) % len(old)
    self.front = 0
    self.back = (self.front + self.size - 1) % len(self._data)

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

Line 27, Column 42

```



```

Oct 25 16:12 • -/gitRepos/Data-Structures-practicals/practical_4.py (Data-Structures-practicals) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
practical_7a.py x practical_7b.py x ds8.py x practical_8.py x practical_3b.py x practical_4.py x

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):
    """Resize to a new list of capacity >= len(self)."""
    old = self._data
    self._data = [None] * cap
    self._data[:len(old)] = old
    self.front = 0
    self.back = (self.front + self.size - 1) % len(self._data)
    walk = self.front
    for k in range(self.size):
        self._data[k] = old[walk] # intentionally shift indices
        walk = (1 + walk) % len(old)
    self.front = 0
    self.back = (self.front + self.size - 1) % len(self._data)

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

(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$ python practical_4.py
First Element: 1, Last Element: 1 105 queue.dequeueEnd()
First Element: 1, Last Element: 2 106 print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
First Element: 2, Last Element: 2 107 queue.enqueueEnd(6)
First Element: 2, Last Element: 3 108 print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
First Element: 2, Last Element: 4 109
First Element: 3, Last Element: 4
First Element: 4, Last Element: 4
First Element: 5, Last Element: 3
First Element: 5, Last Element: 2
First Element: 6, Last Element: 1
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$ 

```

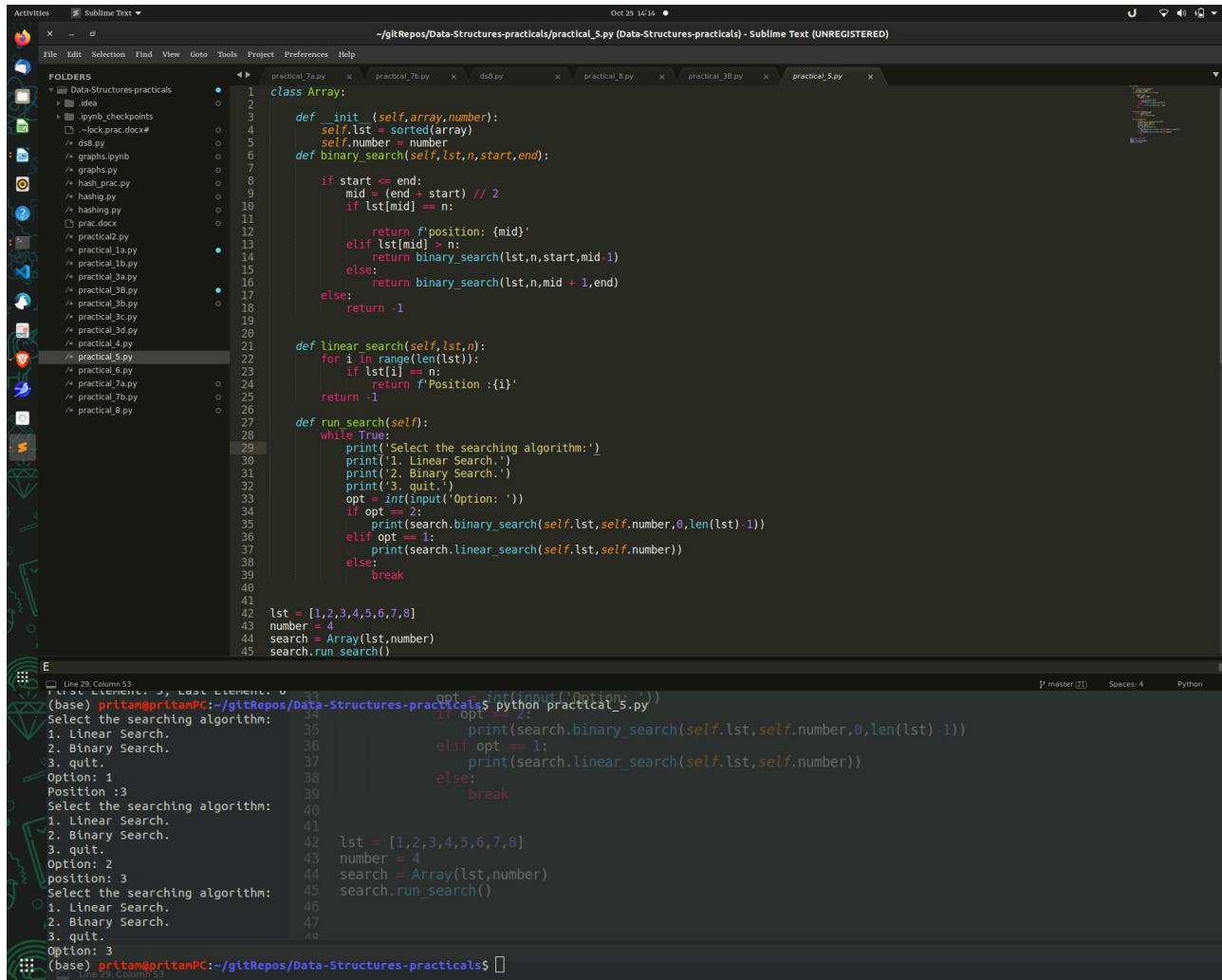
5.

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

Theory

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.

Linear Search: A Linear Search is the most basic type of searching algorithm. A Linear Search sequentially moves through your collection (or data structure) looking for a matching value. In other words, it looks down a list, one item at a time, without jumping.



```

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()

```

Line 29, Column 53
 LIST ELEMENT: 3, LAST ELEMENT: 8
 (base) pritam@pritamPC:/gitRepos/Data-Structures-practicals\$ python practical_5.py
 Select the searching algorithm:
 1. Linear Search.
 2. Binary Search.
 3. quit.
 Option: 1
 Position :3
 Select the searching algorithm:
 1. Linear Search.
 2. Binary Search.
 3. quit.
 Option: 2
 position: 3
 Select the searching algorithm:
 1. Linear Search.
 2. Binary Search.
 3. quit.
 Option: 3
 (base) pritam@pritamPC:/gitRepos/Data-Structures-practicals\$

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

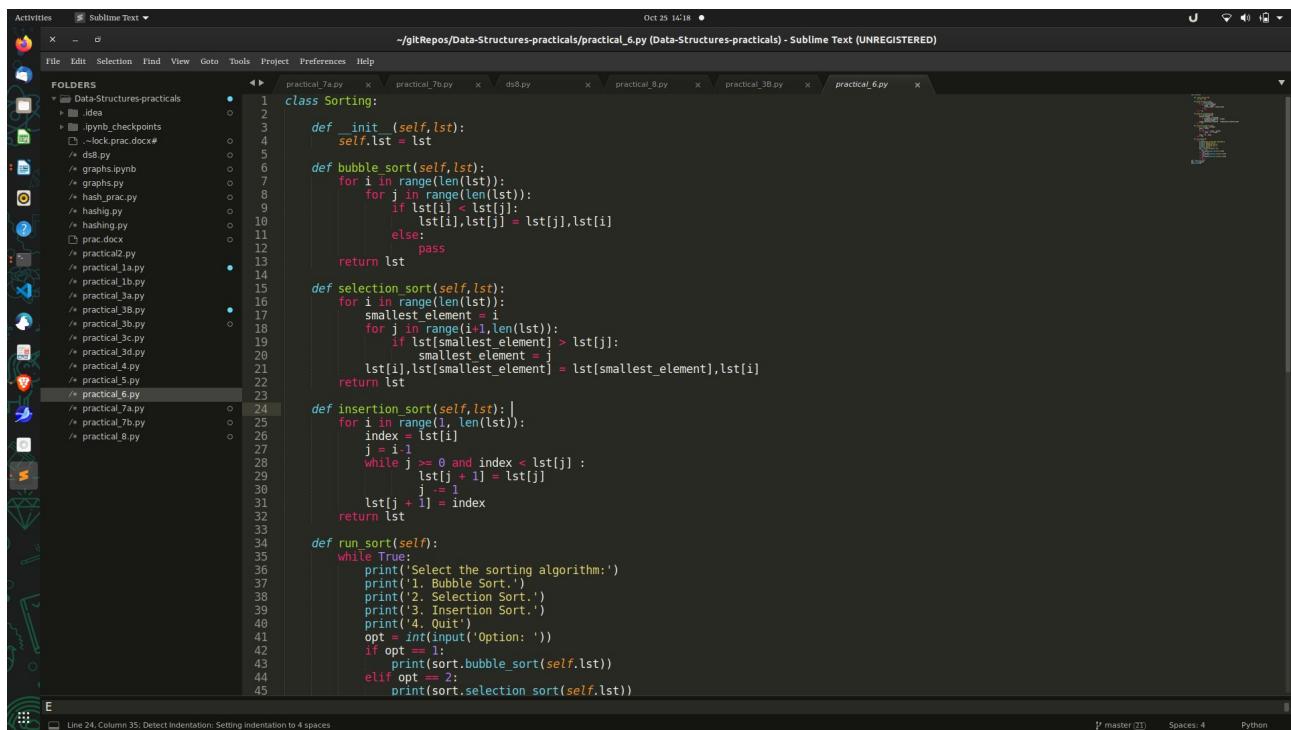
Bubble sort : Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where n is the number of items.

Insertion Sort : This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained

to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.

Selection Sort : Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

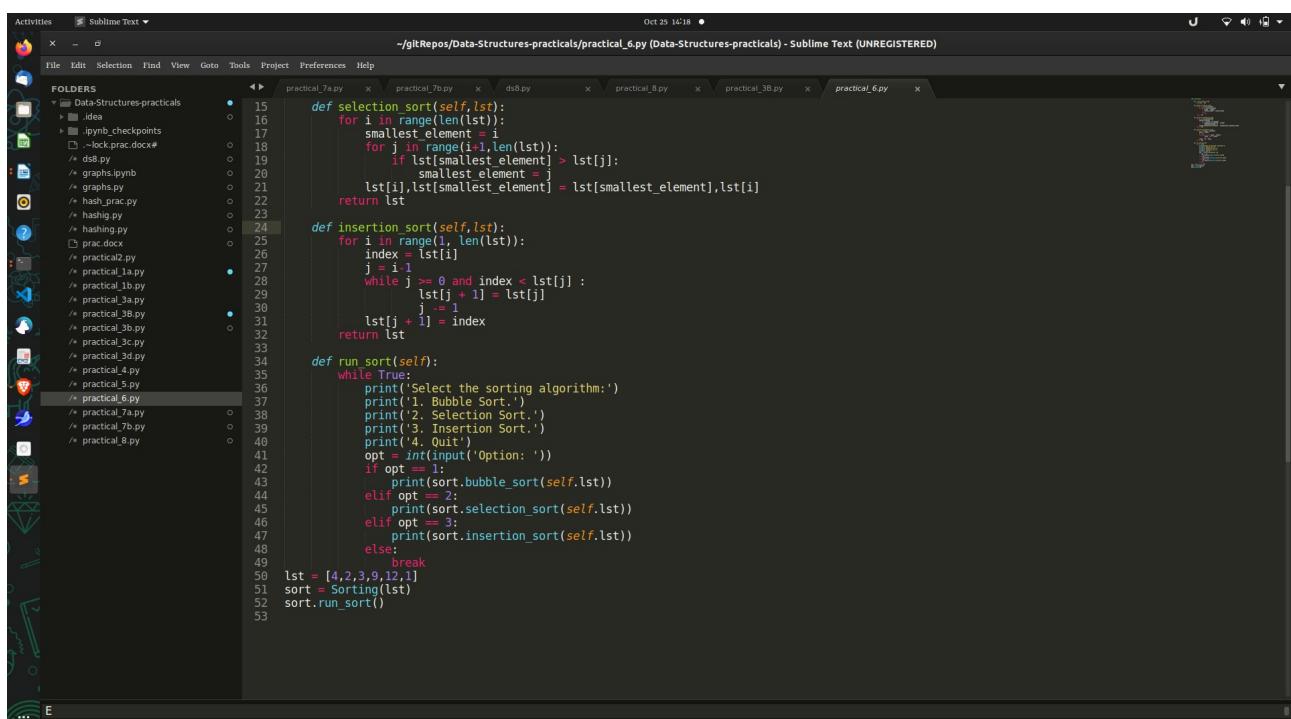
The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.



```

Activities Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
practical_7a.py practical_7b.py ds8.py practical_8.py practical_3B.py practical_6.py
practical_6.py
1 class Sorting:
2     def __init__(self, lst):
3         self.lst = lst
4
5     def bubble_sort(self, lst):
6         for i in range(len(lst)):
7             for j in range(len(lst)):
8                 if lst[i] < lst[j]:
9                     lst[i], lst[j] = lst[j], lst[i]
10                else:
11                    pass
12
13    return lst
14
15    def selection_sort(self, lst):
16        for i in range(len(lst)):
17            smallest_element = i
18            for j in range(i+1, len(lst)):
19                if lst[smallest_element] > lst[j]:
20                    smallest_element = j
21            lst[i], lst[smallest_element] = lst[smallest_element], lst[i]
22
23    return lst
24
25    def insertion_sort(self, lst):
26        for i in range(1, len(lst)):
27            index = lst[i]
28            j = i - 1
29            while j >= 0 and index < lst[j]:
30                lst[j + 1] = lst[j]
31                j -= 1
32            lst[j + 1] = index
33
34    def run_sort(self):
35        while True:
36            print('Select the sorting algorithm:')
37            print('1. Bubble Sort.')
38            print('2. Selection Sort.')
39            print('3. Insertion Sort.')
40            print('4. Quit')
41            opt = int(input('Option: '))
42            if opt == 1:
43                print(sort.bubble_sort(self.lst))
44            elif opt == 2:
45                print(sort.selection_sort(self.lst))
46            elif opt == 3:
47                print(sort.insertion_sort(self.lst))
48            else:
49                break
50
51 lst = [4, 2, 3, 9, 12, 1]
52 sort = Sorting(lst)
53 sort.run_sort()

```



```

Activities Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
practical_7a.py practical_7b.py ds8.py practical_8.py practical_3B.py practical_6.py
practical_6.py
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

```

```
Option: 3
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$ python practical_6.py
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 1
[1, 2, 3, 4, 9, 12]
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 2
[1, 2, 3, 4, 9, 12]
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 3
[1, 2, 3, 4, 9, 12]
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 4
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$ 
```

7. Implement the following for Hashing:

a.

Aim: Write a program to implement the collision technique.

Theory

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

Hashing

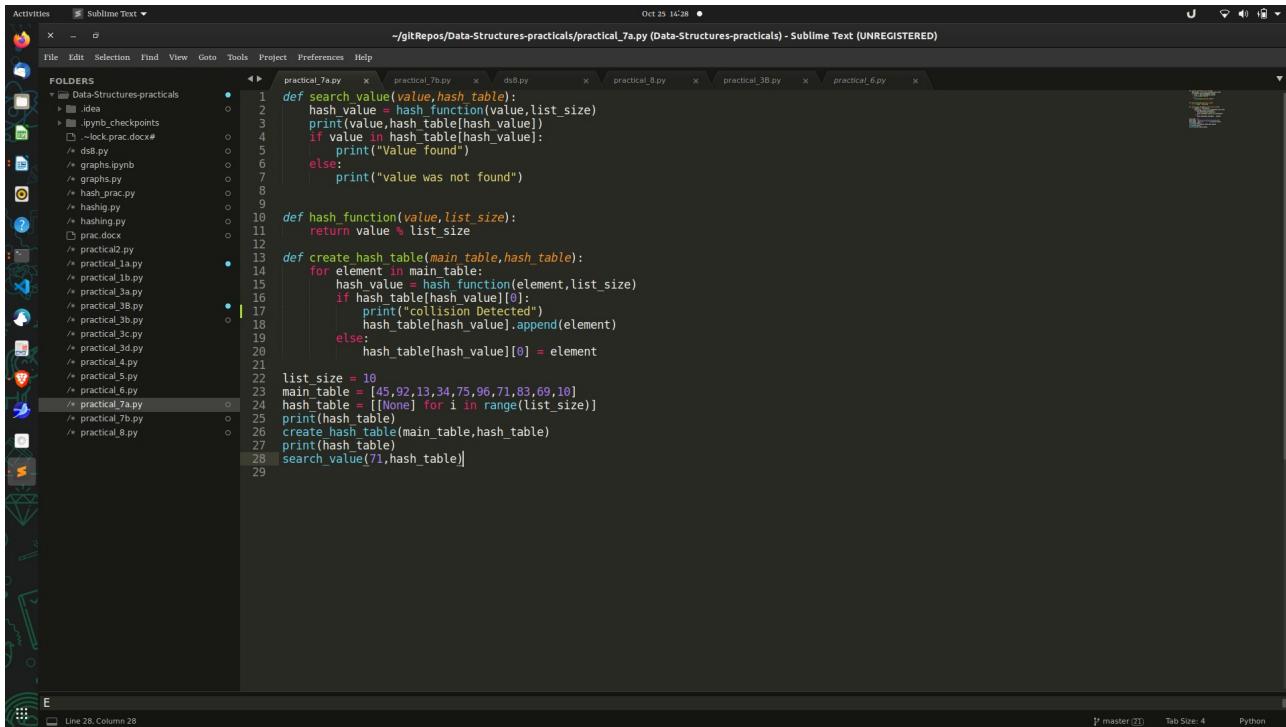
Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.

In computer science, a collision or clash is a situation that occurs when two distinct pieces of data have the same hash value, checksum, fingerprint, or cryptographic digest.[1]

Due to the possible applications of hash functions in data management and computer security (in particular, cryptographic hash functions), collision avoidance has become a fundamental topic in computer science.

Collisions are unavoidable whenever members of a very large set (such as all possible person names, or all possible computer files) are mapped to a relatively short bit string. This is merely an instance of the pigeonhole principle.

The impact of collisions depends on the application. When hash functions and fingerprints are used to identify similar data, such as homologous DNA sequences or similar audio files, the functions are designed so as to maximize the probability of collision between distinct but similar data, using techniques like locality-sensitive hashing. Checksums, on the other hand, are designed to minimize the probability of collisions between similar inputs, without regard for collisions between very different inputs.



The screenshot shows a Sublime Text window with multiple tabs open. The active tab contains Python code for a hash table using linear probing. The code defines functions for searching a value, calculating a hash function, creating a hash table, and implementing linear probing. The terminal tab shows the execution of the program, which creates a hash table with 10 slots and inserts values 45, 92, 13, 34, 75, 96, 71, 83, 69, 10. It then searches for the value 71 and finds it at index 7.

```

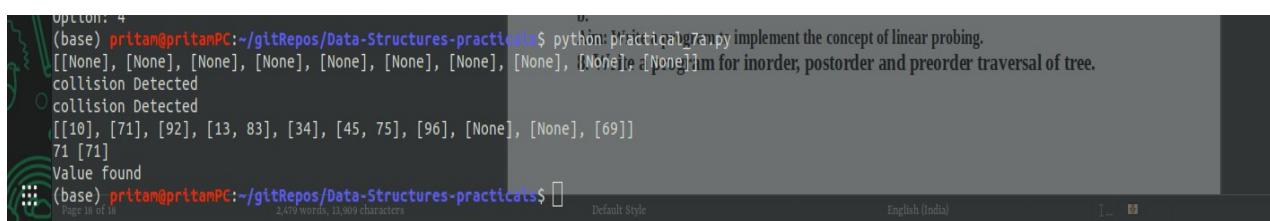
def search_value(value,hash_table):
    hash_value = hash_function(value,list_size)
    print(value,hash_table[hash_value])
    if value in hash_table[hash_value]:
        print("Value found")
    else:
        print("value was not found")

def hash_function(value,list_size):
    return value % list_size

def create_hash_table(main_table,hash_table):
    for element in main_table:
        hash_value = hash_function(element,list_size)
        if hash_table[hash_value][0]:
            print("Collision Detected")
            hash_table[hash_value].append(element)
        else:
            hash_table[hash_value][0] = element

list_size = 10
main_table = [45,92,13,34,75,96,71,83,69,10]
hash_table = [[None] for i in range(list_size)]
print(hash_table)
create_hash_table(main_table,hash_table)
print(hash_table)
search_value(71,hash_table)

```



The terminal window shows the execution of the Python script. It prints the main table and the hash table, then searches for the value 71, finding it at index 7. The output is as follows:

```

option: 4
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$ python practical_7a.py implement the concept of linear probing.
[[None], [None], [None], [None], [None], [None], [None], [None], [None], [None]] # comment for inorder, postorder and preorder traversal of tree.
collision Detected
collision Detected
[[10], [71], [92], [13], [83], [34], [45], [75], [96], [None], [None], [69]]
71 [71]
Value found
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practicals$ 

```

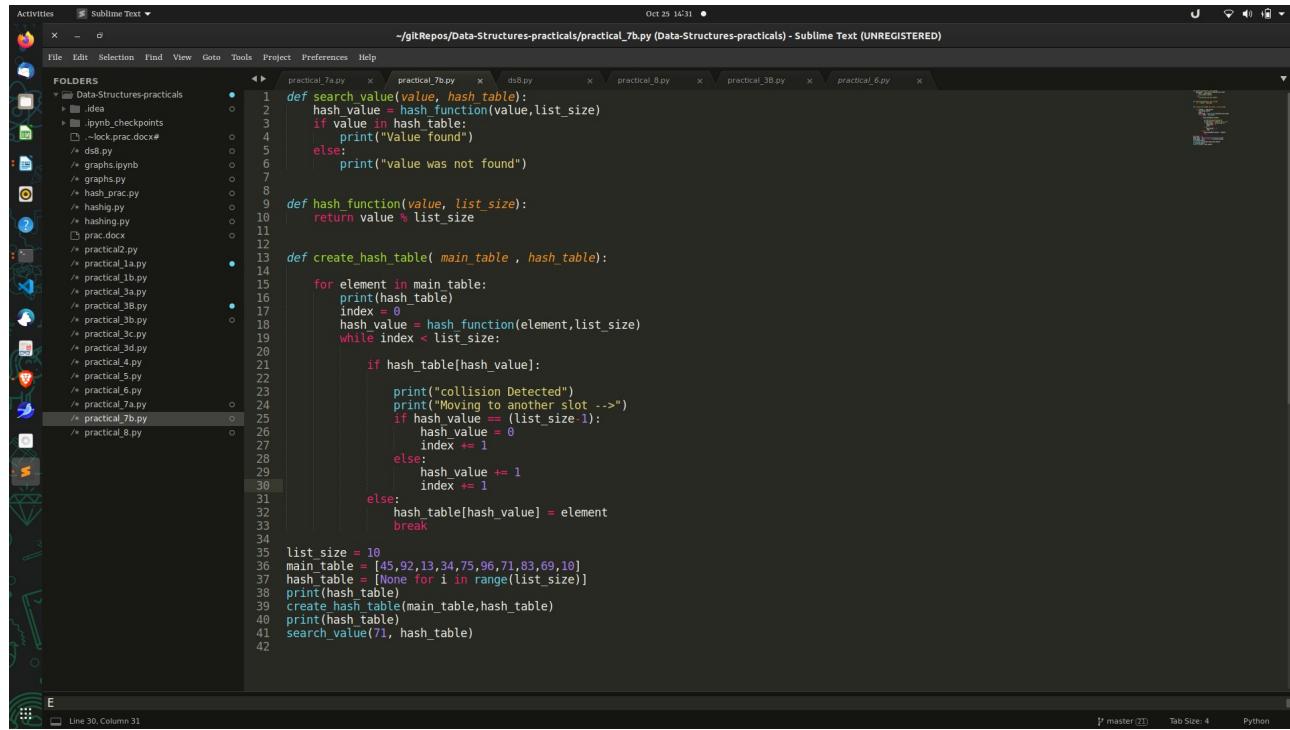
b.

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. It was invented in 1954 by Gene Amdahl, Elaine M. McGraw, and Arthur Samuel and first analyzed in 1963 by Donald Knuth.

Along with quadratic probing and double hashing, linear probing is a form of open addressing. In these schemes, each cell of a hash table stores a single key-value pair. When the hash function causes a collision by mapping a new key to a cell of the hash table that is already occupied by another key, linear probing searches the table for the closest following free location and inserts the new key there. Lookups are performed in the same way, by searching the table sequentially starting at the position given by the hash function, until finding a cell with a matching key or an empty cell.



The screenshot shows a Sublime Text window with multiple tabs open. The active tab contains Python code for a hash table. The code includes functions for searching, calculating hash values, and creating the hash table. It uses a list as the main table and a separate list for the hash table. The code handles collisions by moving elements to other slots if a slot is already occupied.

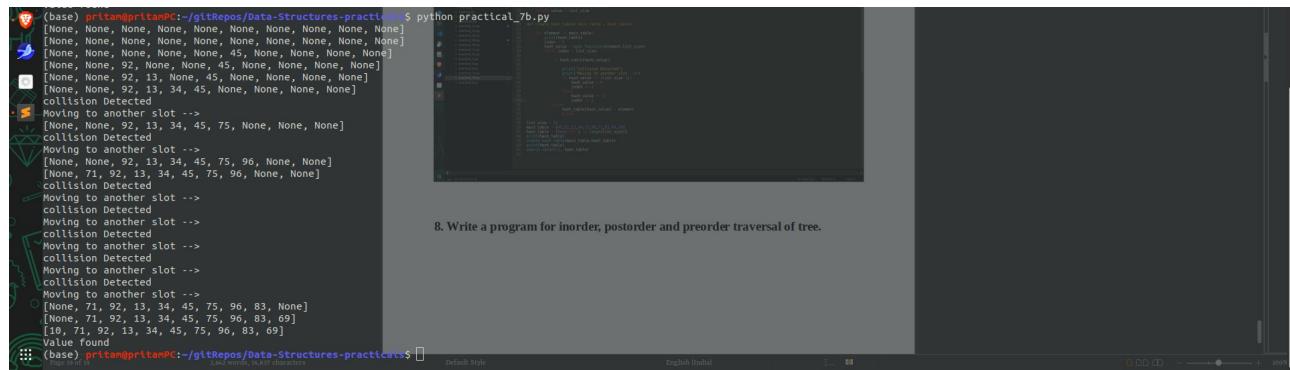
```

def search_value(value, hash_table):
    hash_value = hash_function(value, list_size)
    if value in hash_table:
        print("Value found")
    else:
        print("Value was not found")

def hash_function(value, list_size):
    return value % list_size

def create_hash_table( main_table, hash_table):
    for element in main_table:
        print(hash_table)
        index = 0
        hash_value = hash_function(element, list_size)
        while index < list_size:
            if hash_table[hash_value]:
                print("Collision Detected")
                print("Moving to another slot -->")
                if hash_value == (list_size - 1):
                    hash_value = 0
                else:
                    hash_value += 1
            else:
                hash_table[hash_value] = element
                break
            index += 1
    list_size = 10
    main_table = [45, 92, 13, 34, 75, 96, 71, 83, 69, 10]
    hash_table = [None for i in range(list_size)]
    print(hash_table)
    create_hash_table(main_table,hash_table)
    print(hash_table)
    search_value(71, hash_table)

```



The screenshot shows a terminal window running on a Mac OS X system. The user has run the command `python practical_7b.py`. The output shows the creation of a hash table from a main table of integers. It handles collisions by moving elements to other slots. The final state of the hash table is printed, showing that the value 71 was found at index 0.

```

(base) pritam@pritamPC:~/gitRepos/Data-Structures-practical$ python practical_7b.py
[None, None, None, None, None, None, None, None, None]
[None, None, None, None, None, None, None, None, None]
[None, None, None, None, None, None, None, None, None]
[None, None, 92, None, None, 45, None, None, None]
[None, 71, 92, 13, None, 45, None, None, None]
[None, None, 92, 13, 34, 45, None, None, None]
collision Detected
Moving to another slot -->
[None, None, 92, 13, 34, 45, 75, None, None, None]
collision Detected
Moving to another slot -->
[None, None, 92, 13, 34, 45, 75, 96, None, None]
collision Detected
Moving to another slot -->
[None, 71, 92, 13, 34, 45, 75, 96, 83, None]
collision Detected
Moving to another slot -->
[None, 71, 92, 13, 34, 45, 75, 96, 83, 69]
[10, None, 92, 13, 34, 45, 75, 96, 83, 69]
Value Found
Value found
(base) pritam@pritamPC:~/gitRepos/Data-Structures-practical$
```

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

8.**Aim :Write a program for inorder, postorder and preorder traversal of tree.****Theory****Pre-order (NLR)**

Access the data part of the current node.

Traverse the left subtree by recursively calling the pre-order function.

Traverse the right subtree by recursively calling the pre-order function.

The pre-order traversal is a topologically sorted one, because a parent node is processed before any of its child nodes is done.

In-order (LNR)

Traverse the left subtree by recursively calling the in-order function.

Access the data part of the current node.

Traverse the right subtree by recursively calling the in-order function.

In a binary search tree ordered such that in each node the key is greater than all keys in its left subtree and less than all keys in its right subtree, in-order traversal retrieves the keys in ascending sorted order.[4]

Post-order (LRN)

Traverse the left subtree by recursively calling the post-order function.

Traverse the right subtree by recursively calling the post-order function.

Access the data part of the current node.

The trace of a traversal is called a sequentialisation of the tree. The traversal trace is a list of each visited root. No one sequentialisation according to pre-, in- or post-order describes the underlying tree uniquely. Given a tree with distinct elements, either pre-order or post-order paired with in-order is sufficient to describe the tree uniquely. However, pre-order with post-order leaves some ambiguity in the tree structure.

```

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

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

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

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

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

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

```

The screenshot shows a Sublime Text window with several tabs open, all displaying the same Python file: practical_8.py. The file contains code for a binary search tree. It includes methods for inserting data into the tree based on value (less than or greater than the current node's value), searching for a specific value, and printing the tree in Pre-order, In-order, and Post-order. The code uses a Node class and various helper functions. The terminal below shows the execution of the code and its output.

```
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

def search_value(self, value):
    if self.left:
        self.left.search_value(value)
    if self.value == value:
        print("Value Found")
        return None
    if self.right:
        self.right.search_value(value)
    else:
        print("Value not found")

root = Node(12)
root.insert(13)
root.insert(14)
root.insert(15)
root.insert(16)
root.insert(17)
root.PrintTree()
print("Pre order")
root.Printpreorder()
print("In Order")
root.Printinorder()
print("Post Order")
root.Printpostorder()
root.search_value(17)
```

The terminal window shows the command `python practical_8.py` being run. The output displays the binary tree structure and the results of the search operation.

```
Value Found
```