



**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. RAJ SHAH

Roll No: 360

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	Implement the following for Array: a) Write a program to store the elements in 1-D array and provide an option to perform the operations like searching, sorting, merging, reversing the elements. b) Write a program to perform the Matrix addition, Multiplication and Transpose Operation.	
2	11/09/2020	Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.	
3	18/09/2020	Implement the following for Stack: a) Perform Stack operations using Array implementation. b. b) Implement Tower of Hanoi. c) WAP to scan a polynomial using linked list and add two polynomials. d) WAP to calculate factorial and to compute the factors of a given no. (i) using recursion, (ii) using iteration	
4	25/09/2020	Perform Queues operations using Circular Array implementation.	
5	01/10/2020	Write a program to search an element from a list. Give user the option to perform Linear or Binary search.	
6	09/10/2020	WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.	
7	16/10/2020	Implement the following for Hashing: a) Write a program to implement the collision technique. b) Write a program to implement the concept of linear probing.	
8	23/10/2020	Write a program for inorder, postorder and preorder traversal of tree.	

# Practical 1(a)

Aim: 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.

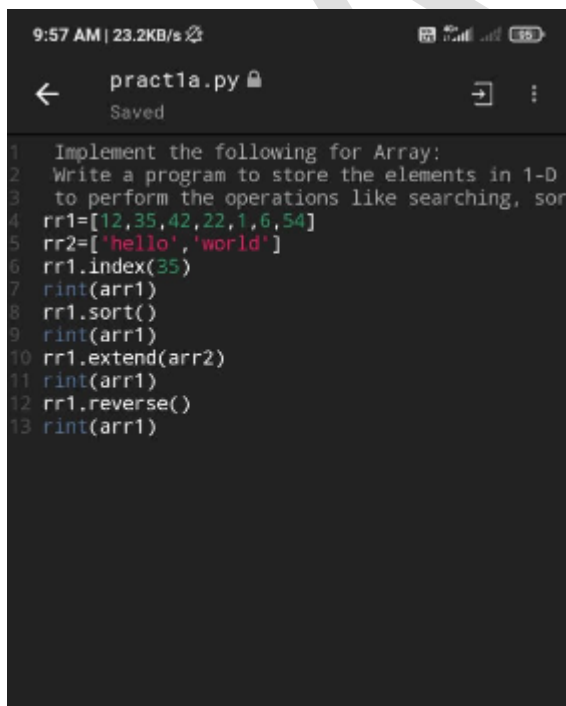
Theory:

Storing Data in Arrays. Assigning values to an element in an array is similar to assigning values to scalar variables. Simply reference an individual element of an array 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.

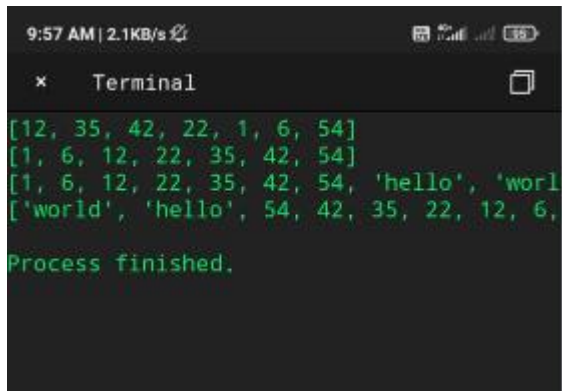
- Traverse – print all the array elements one by one.
- Insertion – Adds an element at the given index.
- Deletion – Deletes an element at the given index.
- Search – Searches an element using the given index or by the value

Code:

A screenshot of a code editor window titled 'pract1a.py' with a 'Saved' status. The editor shows a Python script with 13 lines of code. The code implements a 1D array and performs various operations: initialization, index finding, printing, sorting, extending, and reversing. The background of the code editor has a large, faint watermark that reads 'W3Schools'.

```
1 Implement the following for Array:
2 Write a program to store the elements in 1-D
3 to perform the operations like searching, sor
4 rr1=[12,35,42,22,1,6,54]
5 rr2=['hello','world']
6 rr1.index(35)
7 rint(arr1)
8 rr1.sort()
9 rint(arr1)
10 rr1.extend(arr2)
11 rint(arr1)
12 rr1.reverse()
13 rint(arr1)
```

Output:

A screenshot of a mobile terminal window. The status bar at the top shows the time as 9:57 AM, a data speed of 2.1KB/s, and various icons for signal, Wi-Fi, and battery. The terminal window has a title bar with a close button, the word "Terminal", and a window icon. The output text is as follows:

```
[12, 35, 42, 22, 1, 6, 54]
[1, 6, 12, 22, 35, 42, 54]
[1, 6, 12, 22, 35, 42, 54, 'hello', 'world']
['world', 'hello', 54, 42, 35, 22, 12, 6,
Process finished.
```

GitHub Link:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/Prac1a>

# Practical 1(b)

Aim: Implement the following for Array:

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.

Code:

```
9:57 AM | 16.8KB/s
← prac1b.py Saved
# Program to add two matrices
X = [[1,7,3],
      [4,5,6],
      [7,8,9]]

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

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

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

# Program to multiply two matrices
# 3x3 matrix
X = [[12,7,3],
      [4,5,6],
      [7,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
      [6,7,3,0],
      [4,5,9,1]]
# result is 3x4
result = [[0,0,0,0],
          [0,0,0,0],
          [0,0,0,0]]
# iterate through rows of X
for i in range(len(X)):
    # iterate through columns of Y
    for j in range(len(Y[0])):
        # iterate through rows of Y
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]
        for r in result:
            print(r)

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

Output:

```
9:57 AM | 2.5KB/s
Terminal
[74, 97, 73, 14]
[119, 157, 0, 0]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 7, 0]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 31, 0]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 0]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 14]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 14]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
[12, 0, 0]
[0, 0, 0]
[12, 0, 0]
[7, 0, 0]
[12, 4, 0]
[7, 0, 0]
[12, 4, 0]
[7, 5, 0]
[12, 4, 3]
[7, 5, 0]
[12, 4, 3]
[7, 5, 8]
Process finished.
```

GitHub Link:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/Prac1b%20matrices>

# Practical 2

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

**Theory:**

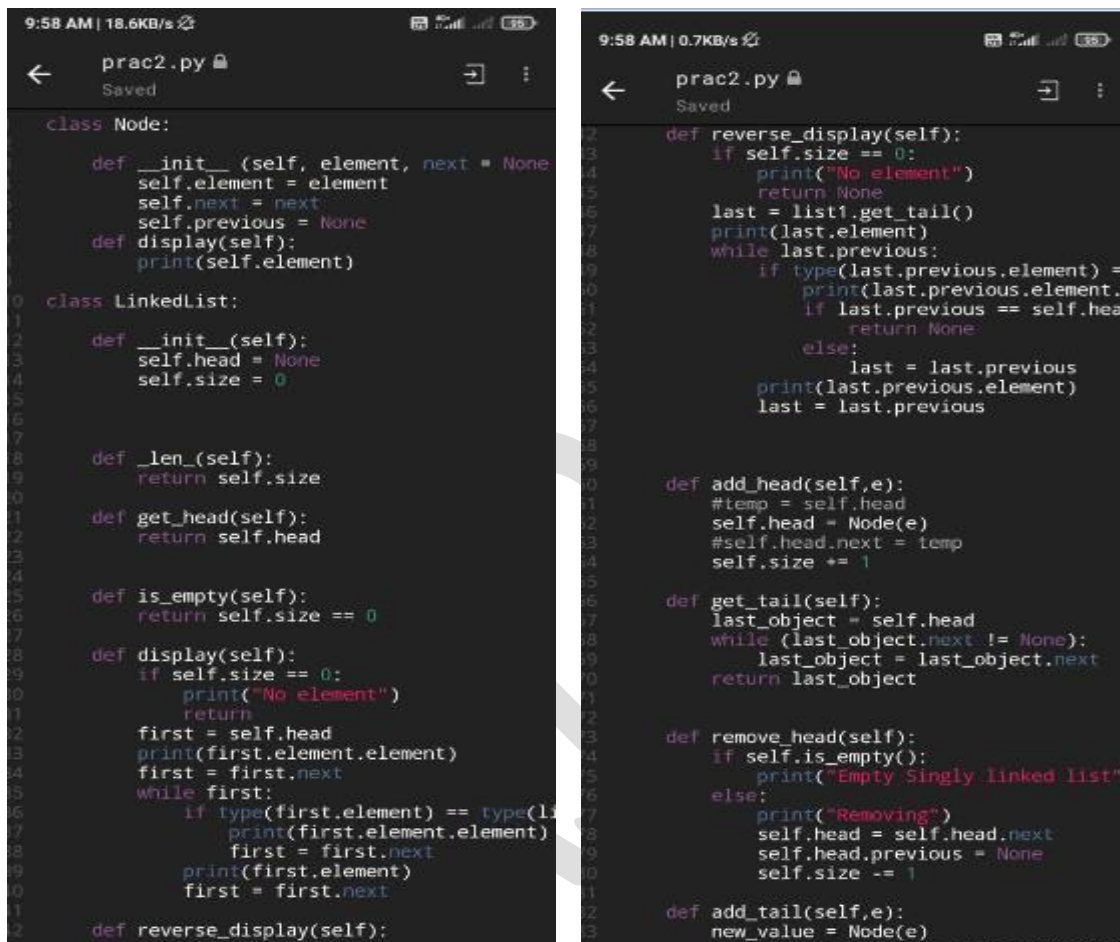
A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as 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 Linked List recursively we need to divide the Linked List into two parts: head and remaining. Head points to the first element initially. Remaining points to the next element from the head. We traverse the Linked List 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

Code:



```

9:58 AM | 18.6KB/s
← prac2.py Saved
class Node:
    def __init__(self, element, next = None, previous = None):
        self.element = element
        self.next = next
        self.previous = previous
    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)
        first = first.next
        while first:
            if type(first.element) == type(1):
                print(first.element)
                first = first.next
            else:
                print(first.element)
                first = first.next

    def reverse_display(self):
        if self.size == 0:
            print("No element")
            return None
        last = self.get_tail()
        print(last.element)
        while last.previous:
            if type(last.previous.element) == type(1):
                print(last.previous.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()

```

```

9:58 AM | 0.7KB/s
← prac2.py Saved
135 self.get_tail().next = new_value
136 self.size += 1
137
138 def find_second_last_element(self):
139     #second_last_element = None
140
141     if self.size >= 2:
142         first = self.head
143         temp_counter = self.size - 2
144         while temp_counter > 0:
145             first = first.next
146             temp_counter -= 1
147         return first
148
149     else:
150         print("Size not sufficient")
151
152     return None
153
154 def remove_tail(self):
155     if self.is_empty():
156         print("Empty Singly linked list")
157     elif self.size == 1:
158         self.head == None
159         self.size -= 1
160     else:
161         Node = self.find_second_last_element()
162         if Node:
163             Node.next = None
164             self.size -= 1
165
166 def get_node_at(self, index):
167     element_node = self.head
168     counter = 0
169     if index == 0:
170         return element_node.element
171     if index > self.size-1:
172         print("Index out of bound")
173     return None

```

```

9:58 AM | 0.7KB/s
← prac2.py Saved
157 elif position == self.size:
158     self.add_tail(element)
159 elif position == 0:
160     self.add_head(element)
161 else:
162     prev_node = self.get_node_at(position)
163     current_node = self.get_node_at(position+1)
164     prev_node.next = element_node
165     element_node.previous = prev_node
166     element_node.next = current_node
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 type(value.element) == type(linkedlist_value.element):
175             print("Searching at " + str(index))
176             if value.element == search_value:
177                 print("Found value at " + str(index))
178                 return True
179             index += 1
180         else:
181             print("Not Found")
182             return False
183
184 def merge(self, linkedlist_value):
185     if self.size > 0:
186         last_node = self.get_node_at(self.size-1)
187         last_node.next = linkedlist_value.head
188         linkedlist_value.head.previous = last_node
189         self.size = self.size + linkedlist_value.size
190     else:
191         self.head = linkedlist_value.head
192         self.size = linkedlist_value.size
193
194 l1 = Node('element 1')
195 list1 = LinkedList()
196 list1.add_head(l1)
197 list1.add_tail('element 2')

```

```

9:58 AM | 0.9KB/s
← prac2.py Saved
75     print("Searching at " + str(index))
76     else:
77         print("Searching at " + str(index))
78         if value.element == search_value:
79             print("Found value at " + str(index))
80             return True
81         index += 1
82     print("Not Found")
83     return False
84
85     def merge(self, linkedlist_value):
86         if self.size > 0:
87             last_node = self.get_node_at(self.size - 1)
88             last_node.next = linkedlist_value.head
89             linkedlist_value.head.previous = last_node
90             self.size = self.size + linkedlist_value.size
91         else:
92             self.head = linkedlist_value.head
93             self.size = linkedlist_value.size
94
95     l1 = Node('element 1')
96     list1 = LinkedList()
97     list1.add_head(l1)
98     list1.add_tail('element 2')
99     list1.add_tail('element 3')
100    list1.add_tail('element 4')
101    list1.get_head().element.element
102    list1.add_between_list(2, 'element between')
103    list1.remove_between_list(2)
104
105    list2 = LinkedList()
106    l2 = Node('element 5')
107    list2.add_head(l2)
108    list2.add_tail('element 6')
109    list2.add_tail('element 7')
110    list2.add_tail('element 8')
111    list1.merge(list2)
112    list1.get_previous_node_at(3).element
113    list1.reverse_display()
114    list1.search('element 6')

```

Output:

```

9:58 AM | 9.2KB/s
× Terminal
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

Process finished.

```

GitHub Link:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac2%20linked%20list>

## Practical 3(a)

Aim: Implement the following for Stack:

- a) 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 objects that 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

Code:

```
9:58 AM | 17.6KB/s
← peac3a.py Saved →
1 class Stack:
2     def __init__(self):
3         self.stack=[]
4
5     def add(self,data):
6
7         if data not in self.stack:
8             self.stack.append(data)
9             return True
10        else:
11            return False
12
13    def top(self):
14        return self.stack[-1]
15
16    def remove(self):
17        if len(self.stack)<=0:
18            return ("No element in Stack")
19
20        else:
21            return self.stack.pop()
22
23 B=Stack()
24 B.add('E')
25 B.add('M')
26 B.top()
27 B.add('J')
28 B.add('R')
29 print(B.top())
30 print(B.remove())
31 print(B.remove())
```

Output:

```
9:58 AM | 2.3KB/s
× Terminal
R
R
J
Process finished.
```

GitHub:

[https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/PRAC3\(A\)%20stack](https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/PRAC3(A)%20stack)

## Practical 3(b)

Aim: Implement Tower of Hanoi.

Theory:

- We are given  $n$  disks and a series of rods, we need to transfer all the disks to the final rod under the given constraints
- We can move only one disk at a time.
- Only the uppermost disk.

Code:

```
9:58 AM | 3.4KB/s
← prac3b.py Saved
1 Recursive Python function to solve the tower
2
3
4
5 def TowerOfHanoi(n , source, destination, auxil
6
7     if n==1:
8         print "Move disk 1 from source",source,
9
10        print "Move disk",n,"from source",source,"t
11
12        TowerOfHanoi(n-1, source, auxiliary, destin
13
14        print "Move disk",n,"from source",source,"t
15
16        TowerOfHanoi(n-1, auxiliary, destination, s
17
18
19
20 Driver code
21
22 n = 4
23
24 TowerOfHanoi(n,'A','B','C')
25 A, C, B are the name of rods
```

Output:

```
9:59 AM | 3.8KB/s
Terminal
Move disk 1 from source A to destination
Move disk 2 from source A to destination
Move disk 1 from source C to destination
Move disk 3 from source A to destination
Move disk 1 from source B to destination
Move disk 2 from source B to destination
Move disk 1 from source A to destination
Move disk 4 from source A to destination
Move disk 1 from source C to destination
Move disk 2 from source C to destination
Move disk 1 from source B to destination
Move disk 3 from source C to destination
Move disk 1 from source A to destination
Move disk 2 from source A to destination
Move disk 1 from source C to destination

Process finished.
```

GitHub:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac3b%20tower%20of%20hanoi>



# Practical 3(C)

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 :  $4x^7 + 12x^2 + 45$

Code:

```
2:44 PM | 0.3KB/s | Saved
← prac3. 3.py
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 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
```

Output:

```
File "source_file.py", line 50
    my_list.add_head(Node(int(input(f"Ent
SyntaxError: invalid syntax
Process finished with exit code 1.
```

GitHub:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac3c%20polynomial>

# Practical 3(d)

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

(i) using recursion

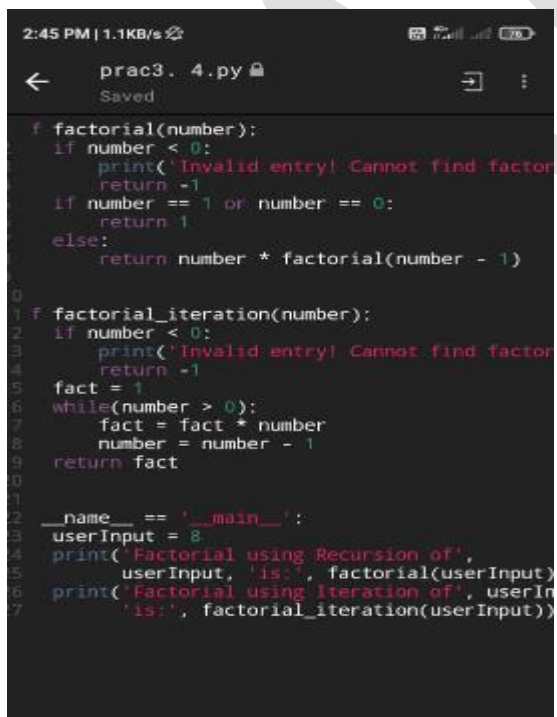
(ii) using iteration

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.

Code:

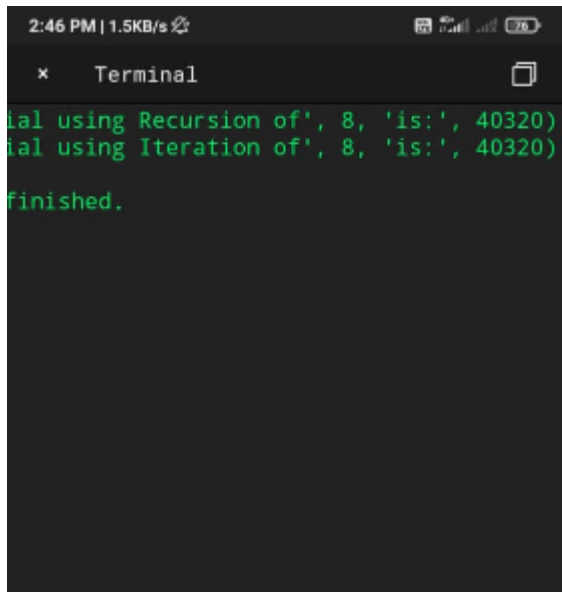


```
2:45 PM | 1.1KB/s
← prac3. 4.py Saved
def factorial(number):
    if number < 0:
        print('Invalid entry! Cannot find factor')
        return -1
    if number == 1 or number == 0:
        return 1
    else:
        return number * factorial(number - 1)

def factorial_iteration(number):
    if number < 0:
        print('Invalid entry! Cannot find factor')
        return -1
    fact = 1
    while(number > 0):
        fact = fact * number
        number = number - 1
    return fact

if __name__ == '__main__':
    userInput = 8
    print('Factorial using Recursion of',
          userInput, 'is:', factorial(userInput))
    print('Factorial using Iteration of', userInput,
          'is:', factorial_iteration(userInput))
```

Output:

A screenshot of a mobile terminal application. The status bar at the top shows the time as 2:46 PM, a data speed of 1.5KB/s, and battery level at 76%. The terminal window has a title bar with a close button and the word "Terminal". The output text is green on a black background. It shows two lines of output: "ial using Recursion of', 8, 'is:', 40320)" and "ial using Iteration of', 8, 'is:', 40320)". The first line is partially cut off. The second line is followed by "finished." on the next line.

```
2:46 PM | 1.5KB/s
× Terminal
ial using Recursion of', 8, 'is:', 40320)
ial using Iteration of', 8, 'is:', 40320)
finished.
```

GitHub:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac3d%20factorial>

# Practical 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 REAR to -1

1. Enqueue Operation check if the queue is full for the first element, set value of FRONT to 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

Code:

```

←  prac4_1.py  Saved
class CircularQueue:
    #Constructor
    def __init__(self):
        self.queue = list()
        self.head = 0
        self.tail = 0
        self.maxSize = 8

    #Adding elements to the queue
    def enqueue(self,data):
        if self.size() == self.maxSize-1:
            return ("Queue Full!")
        self.queue.append(data)
        self.tail = (self.tail + 1) % self.maxSize
        return True

    #Removing elements from the queue
    def dequeue(self):
        if self.size() == 0:
            return ("Queue Empty!")
        data = self.queue[self.head]
        self.head = (self.head + 1) % self.maxSize
        return data

    #Calculating the size of the queue
    def size(self):
        if self.tail >= self.head:
            return (self.tail - self.head)
        return (self.maxSize - (self.head - self.tail))

q = CircularQueue()
print(q.enqueue(1))
print(q.enqueue(2))
print(q.enqueue(3))
print(q.enqueue(4))
print(q.enqueue(5))
print(q.enqueue(6))
print(q.enqueue(7))
print(q.enqueue(8))
print(q.enqueue(9))
print(q.dequeue())
print(q.dequeue())

```

```

42 print(q.dequeue())
43 print(q.dequeue())
44 print(q.dequeue())
45 print(q.dequeue())
46 print(q.dequeue())
47 print(q.dequeue())
48 print(q.dequeue())
49 print(q.dequeue())
50 print(q.dequeue())

```

Try Dcoder's keyboard



Output:

```
× Terminal
True
True
True
True
True
True
True
True
Queue Full!
Queue Full!
1
2
3
4
5
6
7
Queue Empty!
Queue Empty!

Process finished.
```

GitHub:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac4>

# Practical 5

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

Theory:

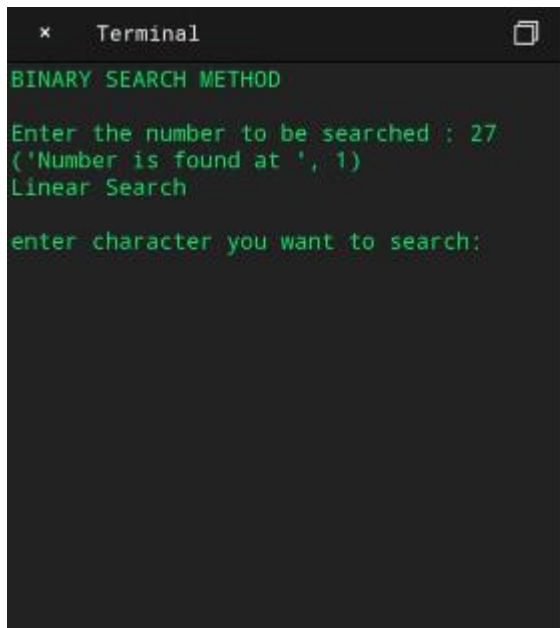
- Linear Search: This linear search is a basic search algorithm which searches all the elements in the list and finds the required value. This is also known as sequential search.
- Binary Search: In computer science, a binary searcher half-interval search algorithm finds the position of a target value within a sorted array. The binary search algorithm can be classified as a dichotomies divide-and-conquer search algorithm and executes in logarithmic time.

Code:

```
← prac5.o.py Saved
print ("BINARY SEARCH METHOD\n")
def bsm(arr,start,end,num):
    if end<=start:
        mid=start+(end-start)//2
        if arr[mid]==x:
            return mid
        elif arr[mid]>x:
            return bsm(arr,start,mid-1,x)
        else:
            return bsm(arr,mid+1,end,x)
    else:
        return -1
arr=[10,27,36,49,58,69,70]
x=int(input("Enter the number to be searched :"))
result=bsm(arr,0,len(arr)-1,x)
if result != -1:
    print ("Number is found at ",result)
else:
    print ("Number is not present\n")

print ("Linear Search\n")
def linearsearch(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1
arr = ['t','u','t','o','r','i','a','l']
x = input("enter character you want to search:")
print("element found at index "+str(linearsearch(arr,x)))
```

Output:

A screenshot of a terminal window titled "Terminal". The text inside is green on a black background. It shows the program's output: "BINARY SEARCH METHOD", "Enter the number to be searched : 27", "('Number is found at ', 1)", "Linear Search", and "enter character you want to search:".

```
× Terminal
BINARY SEARCH METHOD
Enter the number to be searched : 27
('Number is found at ', 1)
Linear Search
enter character you want to search:
```

GitHub:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac5%20linear%20and%20binary%20search>

## Practical 6

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

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 sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two sub arrays 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.

Code:



```
6:37 PM | 11.0KB/s | Saved
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:
                self.bubble_sort(self.lst)
            elif opt == 2:
                self.selection_sort(self.lst)
            elif opt == 3:
                self.insertion_sort(self.lst)
            elif opt == 4:
                break
```

```
6:37 PM | 11.0KB/s | Saved
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         return lst
23
24     def insertion_sort(self, lst):
25         for i in range(1, len(lst)):
26             index = lst[i]
27             j = i-1
28             while j >= 0 and index < lst[j]:
29                 lst[j+1] = lst[j]
30                 j -= 1
31             lst[j+1] = index
32             return lst
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                 self.bubble_sort(self.lst)
44             elif opt == 2:
45                 self.selection_sort(self.lst)
46             elif opt == 3:
47                 self.insertion_sort(self.lst)
48             else:
49                 break
50 lst = [12,4,2,35,43,53,2,5,98,57,87,12]
51 sort = Sorting(lst)
52 sort.run_sort()
```

Output:

```
6:38 PM | 9.0KB/s | [signal icons] [battery icon]
x Terminal [close icon]
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 2
[2, 2, 4, 5, 12, 12, 35, 43, 53, 57, 87, 9
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 1
[2, 2, 4, 5, 12, 12, 35, 43, 53, 57, 87, 9
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 3
[2, 2, 4, 5, 12, 12, 35, 43, 53, 57, 87, 9
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option:
```

GitHub:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/pac6%20option%20to%20select%20the%20sort>

# Practical 7(a)

Aim: Implement the following for Hashing:

Write a program to implement the collision technique.

Theory:

Hashing:

Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.

- **Collisions:** A Hash Collision Attack is an attempt to find two input strings of a hash function that produce the same hash result. If two separate inputs produce the same hash output, it is called a collision.
- **Collision Techniques:** When one or more hash values compete with a single hash table slot, collisions occur. To resolve this, the next available empty slot is assigned to the current hash value
- **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)

```
1:22 PM | 0.7KB/s |   ..  

 prac7. 1.py   

    Saved  

class Hash:  

    def __init__(self, keys, lowerrange, higherrange):  

        self.value = self.hashfunction(keys, lowerrange, higherrange)  

    def get_key_value(self):  

        return self.value  

    def hashfunction(self, keys, lowerrange, higherrange):  

        if lowerrange == 0 and higherrange > 0:  

            return keys%(higherrange)  

if __name__ == '__main__':  

    list_of_keys = [23,43,1,87]  

    list_of_list_index = [None, None, None, None]  

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

    for value in list_of_keys:  

        #print(Hash(value, 0, len(list_of_keys)))  

        list_index = Hash(value, 0, len(list_of_keys)).get_key_value()  

        if list_of_list_index[list_index]:  


            print("Collision detected")  

        else:  

            list_of_list_index[list_index] = value  

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



The screenshot shows a terminal window with a dark background and green text. The window title is "Terminal". The output text is as follows:

```
Before : [None, None, None, None]
Collision detected
Collision detected
After: [None, 1, None, 23]

Process finished.
```

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac7a>

# Practical 7(b)

Aim: Implement the following for Hashing:

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.

Code:

```
1:23 PM 0.0KB/s
← prac7. 2.py Saved → ⋮

1 class Hash:
2     def __init__(self, keys, lowerrange, higherrange):
3         self.value = self.hashfunction(keys, lowerrange, higherrange)
4
5     def get_key_value(self):
6         return self.value
7
8     def hashfunction(self, keys, lowerrange, higherrange):
9         if lowerrange == 0 and higherrange > 0:
10            return keys % higherrange
11
12 if __name__ == '__main__':
13     linear_probing = True
14     list_of_keys = [23, 43, 1, 87]
15     list_of_list_index = [None, None, None, None]
16     print("Before : " + str(list_of_list_index))
17     keys =
18     for value in list_of_keys:
19         list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
20         print("hash value for " + str(value) + " is : " + str(list_index))
21         if list_of_list_index[list_index]:
22             print("Collision detected for " + str(value))
23             if linear_probing:
24                 old_list_index = list_index
25                 if list_index == len(list_of_list_index) - 1:
26                     list_index = 0
27                 else:
28                     list_index += 1
29                 list_full = False
30                 while list_of_list_index[list_index]:
31                     if list_index == old_list_index:
32                         list_full = True
33                     list_index += 1
34                 if list_full:
35                     print("Full hash table")
36                 else:
37                     list_of_list_index[list_index] = value
38                     print("Hashed value for " + str(value) + " is : " + str(list_index))
39             else:
40                 print("Quadratic probing not implemented")
41         else:
42             list_of_list_index[list_index] = value
43             print("Hashed value for " + str(value) + " is : " + str(list_index))
44     print("After : " + str(list_of_list_index))
```

```
1:23 PM 0.5KB/s
← prac7. 2.py Saved → ⋮

16 print("Before : " + str(list_of_list_index))
17 for value in list_of_keys:
18     #print(Hash(value, 0, len(list_of_keys)).get_key_value())
19     list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
20     print("hash value for " + str(value) + " is : " + str(list_index))
21     if list_of_list_index[list_index]:
22         print("Collision detected for " + str(value))
23         if linear_probing:
24             old_list_index = list_index
25             if list_index == len(list_of_list_index) - 1:
26                 list_index = 0
27             else:
28                 list_index += 1
29             list_full = False
30             while list_of_list_index[list_index]:
31                 if list_index == old_list_index:
32                     list_full = True
33                 list_index += 1
34             if list_full:
35                 print("Full hash table")
36             else:
37                 list_of_list_index[list_index] = value
38                 print("Hashed value for " + str(value) + " is : " + str(list_index))
39         else:
40             print("Quadratic probing not implemented")
41     else:
42         list_of_list_index[list_index] = value
43         print("Hashed value for " + str(value) + " is : " + str(list_index))
44     print("After : " + str(list_of_list_index))
```

1:23 PM 0.6KB/s

← prac7. 2.py Saved

```
33         break
34         if list_index+1 == len(list_of_list_index):
35             list_index = 0
36         else:
37             list_index += 1
38     if list_full:
39         print("List was full . Could not save")
40     else:
41         list_of_list_index[list_index] = value
42
43
44
45
46
47
48
```

Try Dcoder's keyboard

1:23 PM 0.6KB/s

← prac7. 2.py Saved

```
44
45
46
47
48
49
50
51     else:
52         list_of_list_index[list_index] = value
53
54     print("After: " + str(list_of_list_index))
```

Try Dcoder's keyboard

Output:

1:23 PM 0.0KB/s

× Terminal

```
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]

Process finished.
```

GitHub:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac7b>

# 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 is 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.

Code:

```
1:54 PM | 1.2KB/s | ...
← prac8.py Saved
1
2 class Node:
3
4     def __init__(self, key):
5
6         self.left = None
7
8         self.right = None
9
10        self.val = key
11
12
13
14
15
16 # A function to do inorder tree traversal
17
18 def printInorder(root):
19
20
21
22     if root:
23
24
25
26         # First recur on left child
27         printInorder(root.left)
28
29
30
31         # then print the data of node
32         print(root.val),
33
34
35
36
37
38         # now recur on right child
39         printInorder(root.right)
40
41
42
43
```

```
1:54 PM | 0.3KB/s | ...
← prac8.py Saved
48 # A function to do postorder tree traversal
49
50 def printPostorder(root):
51
52
53
54     if root:
55
56
57
58         # First recur on left child
59         printPostorder(root.left)
60
61
62
63
64         # the recur on right child
65         printPostorder(root.right)
66
67
68
69         # now print the data of node
70         print(root.val),
71
72
73
74
75
76
77
78 # A function to do preorder tree traversal
79
80 def printPreorder(root):
81
82
83
84     if root:
85
86
87
88         # First print the data of node
89         print(root.val),
90
```

```
1:55 PM | 0.3KB/s | 🔔 🔊 🔋 ...
← prac8.py 🔒 Saved → ⋮

04     # Then recur on left child
05     printPreorder(root.left)
06
07
08     # Finally recur on right child
09     printPreorder(root.right)
10
11
12 # Driver code
13
14 root = Node(1)
15
16 root.left    = Node(2)
17
18 root.right   = Node(3)
19
20 root.left.left = Node(4)
21
22 root.left.right = Node(5)
23
24 print "Preorder traversal of binary tree is"
25
26 printPreorder(root)
27
28 print "\nInorder traversal of binary tree is"
29
30 printInorder(root)
```

Output:



```
1:55 PM | 3.0KB/s | [status icons]
× Terminal [icon]
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Process finished.
```

GitHub:

<https://github.com/RAJ2906/DS-RAJSYIT35/blob/master/prac8>

## Practical 9

Code:

```
2:34 PM | 1.0KB/s | [icons]
← prac 9.py [lock] [share] [menu]
Saved

1
2 # Adjacency Matrix representation in Python
3 class Graph(object):
4 # Initialize the matrix
5 def __init__(self, size): self.adjMatrix = []
6 for i in range(size): self.adjMatrix.append([
7 self.size = size
8 # Add edges
9 def add_edge(self, v1, v2):
10 if v1 == v2: print("Same vertex %d and %d" %
11 self.adjMatrix[v1][v2] = 1 self.adjMatrix[v2][
12 # Remove edges
13 def remove_edge(self, v1, v2):
14 if self.adjMatrix[v1][v2] == 0: print("No edge
15 return self.adjMatrix[v1][v2] = 0
16 self.adjMatrix[v2][v1] = 0
17 def __len__(self):
18 return self.size
19 # Print the matrix
20 def print_matrix(self):
21 for row in self.adjMatrix:
22 for val in row: print('{:4}'.format(val)),
23 print def main():
24 g = Graph(5)
25 g.add_edge(0, 1) g.add_edge(0, 2) g.add_edge(
26 if __name__ == '__main__': main()
```

Output:

```
2:34 PM | 1.6KB/s | [icons]
× Terminal [close]
File "source_file.py", line 3
    class Graph(object):
    ^
IndentationError: unexpected indent
```

## Practical 10

Code:

```
2:33 PM | 0.5KB/s
← prac10.py Saved
1
2
3 import sys
4
5 class Graph():
6
7
8
9     def __init__(self, vertices):
10         self.V = vertices
11         self.graph = [[0 for column in range(vertices)
12                        for row in range(vertices)]]
13
14
15
16
17
18
19     def printSolution(self, dist):
20         print ("Vertex tDistance from Source")
21         for node in range(self.V):
22             print (node, "t", dist[node])
23
24
25
26
27
28
29     # A utility function to find the vertex with
30     # minimum distance value, from the set of
31     # not yet included in shortest path tree
32     def minDistance(self, dist, sptSet):
33
34         # Initilaize minimum distance for next node
35         min = sys.maxsize
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
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
```

Output:

```
2:33 PM | 13.5KB/s
× Terminal
Process finished.
```