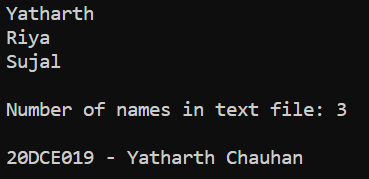
**PRACTICAL – (10.1)**

**AIM: Given a .txt file that has a list of a bunch of names, count how many of each name there are in the file, and print out the results to the screen.**

**PROGRAM:**

|  |
| --- |
| f = open("P\_10.txt", "rt")  data = f.read()  words = data.split()  print("Number of names in text file: ", len(words))  print("\n20DCE019 - Yatharth Chauhan") |

**OUTPUT:**

****

**CONCLUSION:** In this practical, we learned how to count the number of words written in any particular file.

**PRACTICAL – (10.2)**

**AIM: Write a program to implement different Data structures using Python.**

**• Linked List**

**• Stack**

**• Queue**

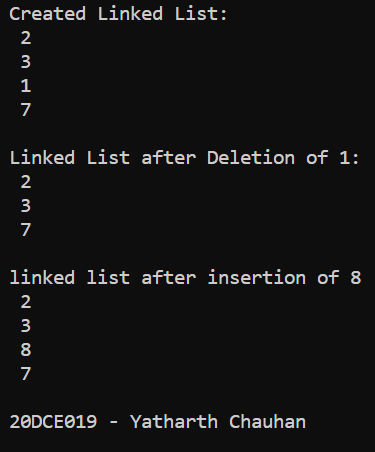
**• Binary Tree**

**PROGRAM:**

1. **Linked List**

|  |
| --- |
| class Node:  def \_\_init\_\_(self, data):  self.data = data  self.next = None  class LinkedList:  def \_\_init\_\_(self):  self.head = None  def push(self, new\_data):  new\_node = Node(new\_data)  new\_node.next = self.head  self.head = new\_node  def deleteNode(self, key):  temp = self.head  if (temp is not None):  if (temp.data == key):  self.head = temp.next  temp = None  return  while(temp is not None):  if temp.data == key:  break  prev = temp  temp = temp.next  if(temp == None):  return  prev.next = temp.next  temp = None  def printList(self):  temp = self.head  while(temp):  print(" %d" % (temp.data)),  temp = temp.next  def insertAfter(self, prev\_node, new\_data):  if prev\_node is None:  print("The given previous node must inLinkedList.")  return  new\_node = Node(new\_data)  new\_node.next = prev\_node.next  prev\_node.next = new\_node  llist = LinkedList()  llist.push(7)  llist.push(1)  llist.push(3)  llist.push(2)  print("Created Linked List: ")  llist.printList()  llist.deleteNode(1)  print("\nLinked List after Deletion of 1:")  llist.printList()  print("\nlinked list after insertion of 8")  llist.insertAfter(llist.head.next, 8)  llist.printList()  print("\n20DCE019 - Yatharth Chauhan") |

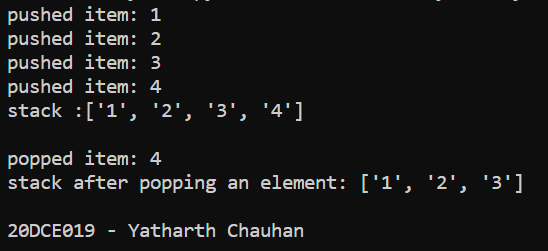
**OUTPUT:**

****

1. **Stack**

|  |
| --- |
| def create\_stack():  stack = []  return stack  def check\_empty(stack):  return len(stack) == 0  def push(stack, item):  stack.append(item)  print("pushed item: " + item)  def pop(stack):  if (check\_empty(stack)):  return "stack is empty"  return stack.pop()  stack = create\_stack()  push(stack, str(1))  push(stack, str(2))  push(stack, str(3))  push(stack, str(4))  print("stack :" + str(stack))  print("\npopped item: " + pop(stack))  print("stack after popping an element: " + str(stack))  print("\n20DCE019 - Yatharth Chauhan") |

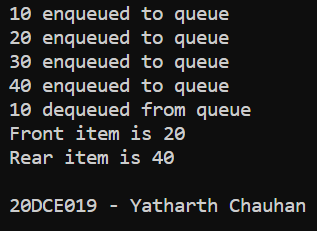
**OUTPUT:**

****

1. **Queue**

|  |
| --- |
| class Queue:  def \_\_init\_\_(self, capacity):  self.front = self.size = 0  self.rear = capacity - 1  self.Q = [None]\*capacity  self.capacity = capacity  def isFull(self):  return self.size == self.capacity  def isEmpty(self):  return self.size == 0  def EnQueue(self, item):  if self.isFull():  print("Full")  return  self.rear = (self.rear + 1) % (self.capacity)  self.Q[self.rear] = item  self.size = self.size + 1  print("% s enqueued to queue" % str(item))  def DeQueue(self):  if self.isEmpty():  print("Empty")  return  print("% s dequeued from queue" % str(self.Q[self.front]))  self.front = (self.front + 1) % (self.capacity)  self.size = self.size - 1  def que\_front(self):  if self.isEmpty():  print("Queue is empty")  print("Front item is", self.Q[self.front])  def que\_rear(self):  if self.isEmpty():  print("Queue is empty")  print("Rear item is", self.Q[self.rear])  if \_\_name\_\_ == '\_\_main\_\_':  queue = Queue(30)  queue.EnQueue(10)  queue.EnQueue(20)  queue.EnQueue(30)  queue.EnQueue(40)  queue.DeQueue()  queue.que\_front()  queue.que\_rear()  print("\n20DCE019 - Yatharth Chauhan") |

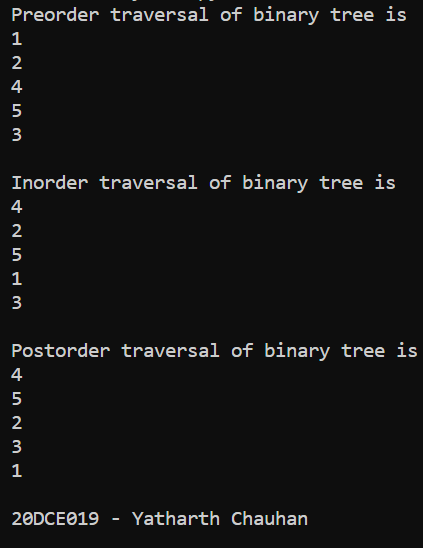
**OUTPUT:**

****

1. **Binary Tree**

|  |
| --- |
| class Node:  def \_\_init\_\_(self, key):  self.left = None  self.right = None  self.val = key  def printInorder(root):  if root:  printInorder(root.left)  print(root.val),  printInorder(root.right)  def printPostorder(root):  if root:  printPostorder(root.left)  printPostorder(root.right)  print(root.val),  def printPreorder(root):  if root:  print(root.val),  printPreorder(root.left)  printPreorder(root.right)  root = Node(1)  root.left = Node(2)  root.right = Node(3)  root.left.left = Node(4)  root.left.right = Node(5)  print("Preorder traversal of binary tree is")  printPreorder(root)  print("\nInorder traversal of binary tree is")  printInorder(root)  print("\nPostorder traversal of binary tree is")  printPostorder(root)  print("\n20DCE019 - Yatharth Chauhan") |

**OUTPUT:**

****

**CONCLUSION:** In this practical we learnt some basic data structures implementation in python for example linked list, stack, queue and binary tree.