JNAN VIKAS MANDAL'S

PADMASHREE DR. R.T.DOSHI DEGREE COLLEGE OF INFORMATION TECHNOLOGY

MOHANLAL RAICHAND MEHTA COLLEGE OF COMMERCE

DIWALIMAA DEGREE COLLEGE OF SCIENCE

AMRATLAL RAICHAND MEHTA COLLEGE OF ARTS

CERTIFICATE

This is to certify that the Mr./Miss. _____ of S.Y.B.Sc.CS Semester III has completed the practical work in the subject of **Data Structure** during the Academic year 2023-24 under the guidance of **Dr. Sanjivani Nalkar** being the partial requirement for the fulfilment of the curriculum of Degree of Bachelor of Science in Computer Science, University of Mumbai.

Place: **Date:**

---------- ----------

Sign of Subject In Charge Sign of External Examiner

Sign of Incharge / H.O.D

INDEX

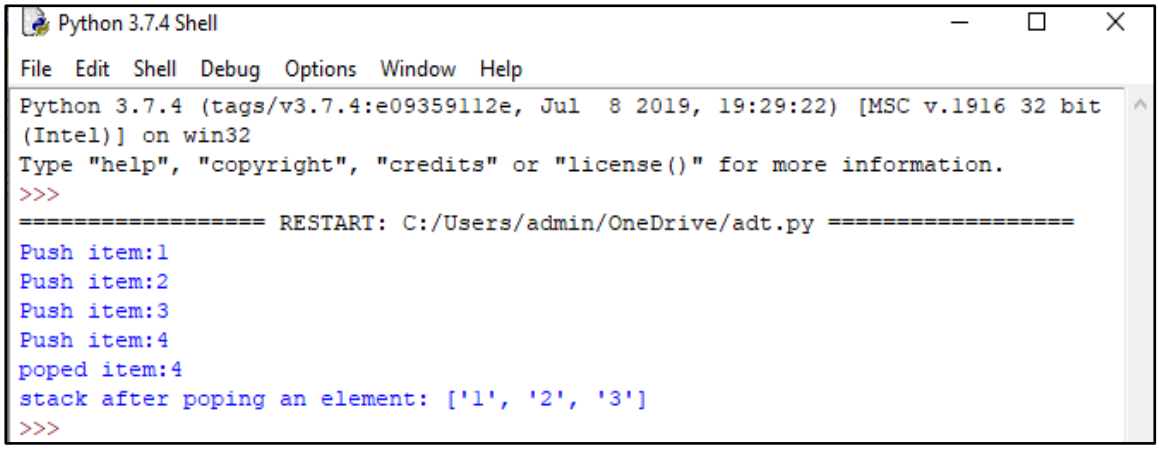
Sr.No.	Name Of Practical	Date	Signature
1	Write a program to implement Abstract Data Types(ADT)	12/08/2023	
2	Write a program to implement stack with insertion, deletion, traversal operation	19/08/2023	
3	Write a program to implement queue with insertion, deletion, traversal operations	09/09/2023	
4	Write a program to implement binary tree with insertion, deletion, traversal operations	16/09/2023	
5	Write a program to implement graph with insertion, deletion, traversal operations	30/09/2023	
6	Write a program to create basic hash table for insertion, deletion, traversal, operation(Assume there is no collision)	30/09/2023	
7	Write a program to create hash table to handle collisions using overflow chaining	07/10/2023	
8	Write a program to implement priority queue with insertion, deletion, traversal operations	07/10/2023	

Practical 1: Write a program to implement Abstract Data Types(ADT)

```
#Stack implementation in python

#Creating stack in python
def create_stack():
        Stack = []
        Return stack

#Creating an empty stack
def check_empty(stack):
        return len(stack) = = 0

#Adding items into stack
def push(stack, item):
        stack.append(item)
        print("pushed item:" + item)

#Removing an element from stack
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("poped item:" + pop(stack))
```

print(“stack after poping an element: “ + str(stack))

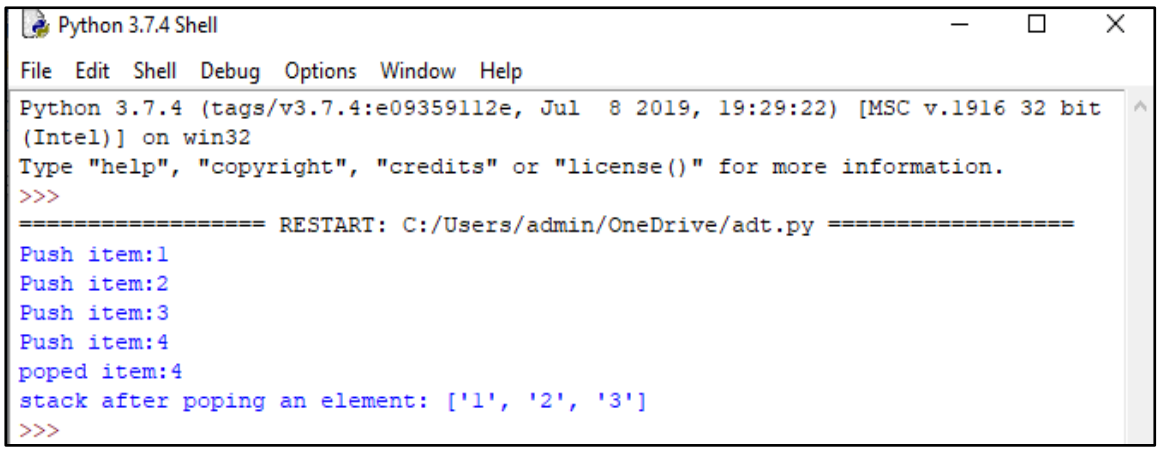
OUTPUT:-



Practical 2: Write a program to implement stack with insertion, deletion, traversal operation

```
# stack implementation in python
#Creating Stack
def create_stack():
        stack = []
        return stack
#Creting an empty stack
def check_empty(stack):
        return len(stack) == 0
#Adding items into stack
def push(stack,item):
        stack.append(item)
        print(“push item:” + item)
#Removing an element from stack
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(“popped items: “ + pop(stack))
print(“stack after poping an element: “ + str(stack))
```

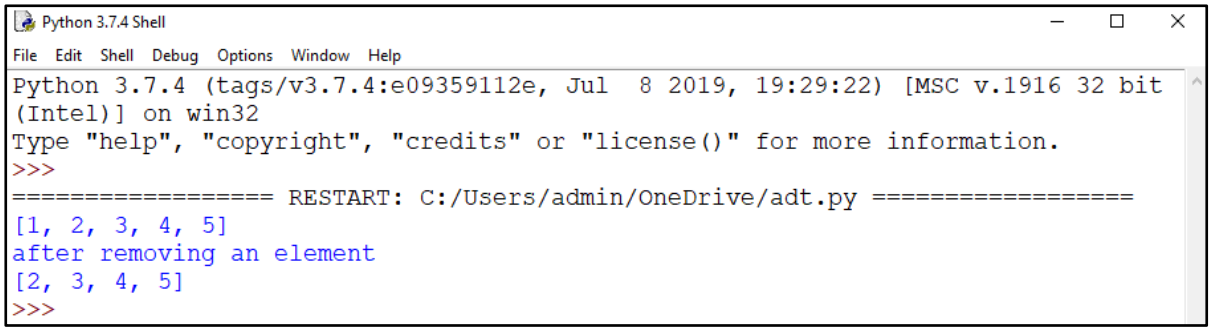
OUTPUT:


Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/admin/OneDrive/adt.py ==========
Push item:1
Push item:2
Push item:3
Push item:4
poped item:4
stack after poping an element: ['1', '2', '3']
>>>

Practical 3: Write a program to implement queue with insertion, deletion, traversal operations

```
# Queue implementation in python
class Queue:
    def __init__(self):
        self.queue = []
#Add an element
def enqueue(self, item):
        self.queue.append(item)
#Remove an element
def dequeue(self):
    if len(self.queue)<1:
        return None
    return self.queue.pop(0)
#Display the queue
def display(self):
    print(self.queue)
def size(self):
    return len(self.queue)
q = Queue()
enqueue(q, 1)
enqueue(q, 2)
enqueue(q, 3)
enqueue(q, 4)
enqueue(q, 5)
display(q)
dequeue(q)
print('after removing an element')
display(q)
```

OUTPUT:

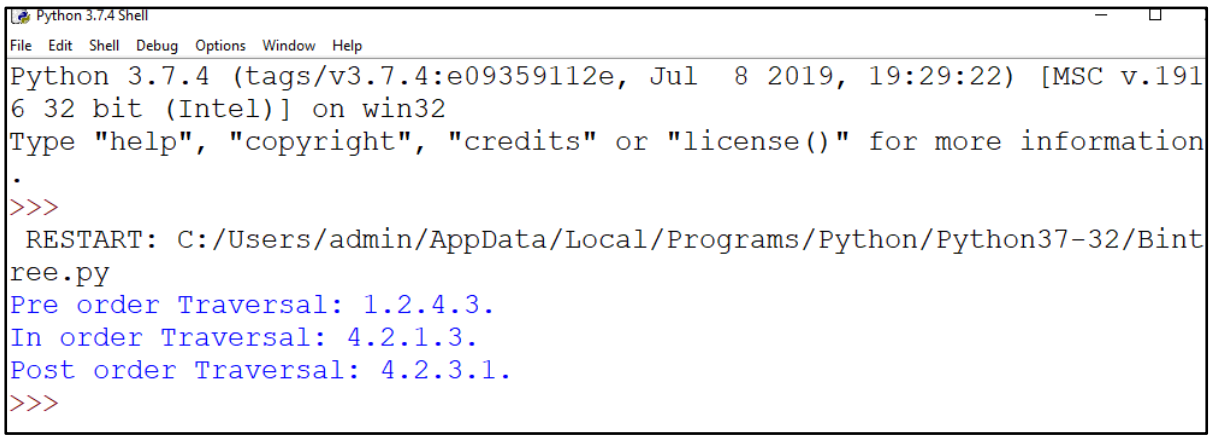

Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/admin/OneDrive/adt.py ==========
[1, 2, 3, 4, 5]
after removing an element
[2, 3, 4, 5]
>>>

Practical 4: Write a program to implement binary tree with insertion, deletion, traversal operations

```
#Binary tree in python


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


# Traverse preorder
    def traversePreOrder(self):
        print(self.val,end='.')
        if self.left:
            self.left.traversePreOrder()
        if self.right:
            self.right.traversePreOrder()


#traverse inorder
    def traverseInOrder(self):
        if self.left:
            self.left.traverseInOrder()
        print(self.val,end='.')
        if self.right:
            self.right.traverseInOrder()
#Traverse postorder
    def traversePostOrder(self):
        if self.left:
```

```
            self.left.traversePostOrder()
        if self.right:
            self.right.traversePostOrder()
        print(self.val,end='.')
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
print("Pre order Traversal:", end=" ")
root.traversePreOrder()
print("\nIn order Traversal:", end=" ")
root.traverseInOrder()
print("\nPost order Traversal:", end=" ")
root.traversePostOrder()
```

OUTPUT:

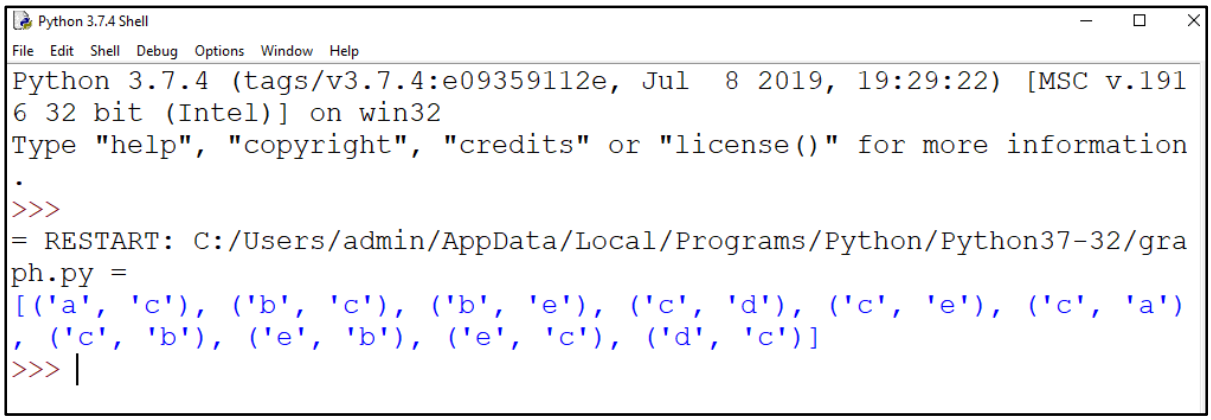


Practical 5: Write a program to implement graph with insertion, deletion, traversal operations

```
#Python program for
#validation of a graph

#import dictionary for graph
from collections import defaultdict

#function for adding edge to graph
graph = defaultdict(list)
def addEdge(graph, u,v):
    graph[u].append(v)

#definition of function
def generate_edge(graph):
    edges = []
    #for each node in graph
    for node in graph:
        #for each neighbour node of a single node
        for neighbour in graph[node]:
            #if edge exists then append
            edges.append((node,neighbour))
    return edges

#declaration of graph as dictonary
addEdge(graph,'a','c')
```

```
addEdge(graph,'b','c')
addEdge(graph,'b','e')
addEdge(graph,'c','d')
addEdge(graph,'c','e')
addEdge(graph,'c','a')
addEdge(graph,'c','b')
addEdge(graph,'e','b')
addEdge(graph,'d','c')
addEdge(graph,'e','c')

#Driver Function call to print generated graph
print(generate_edge(graph))
```

OUTPUT:

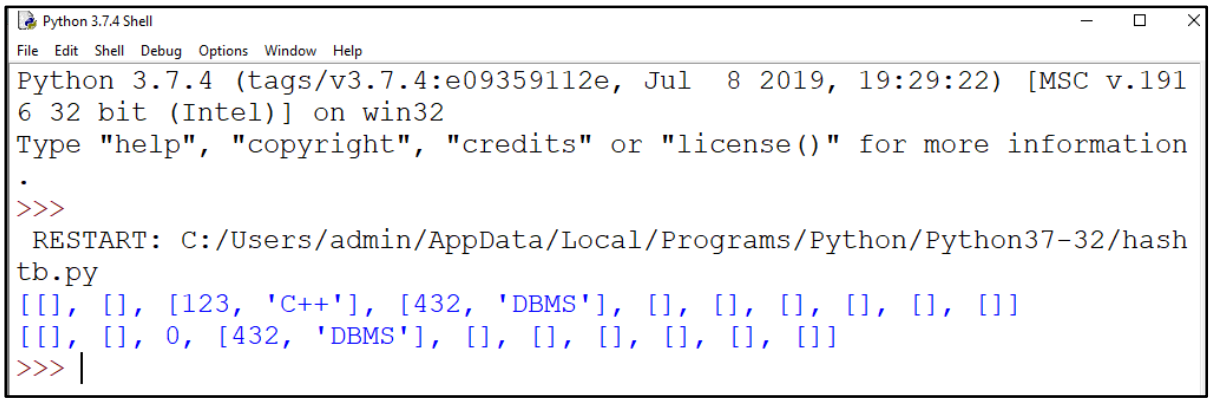


Practical 6:Write a program to create basic hash table for insertion, deletion, traversal, operation(Assume there is no collision)

```
#Python program to demonstrate working of hashtable


hashTable = [[],]*10


def checkPrime(n):
    if n==1 or n==0:
        return 0


    for i in range(2,n//2):
        if n % i == 0:
            return 0


    return 1


def getPrime(n):
    if n % 2 == 0:
        n = n+1
    while not checkPrime(n):
        n += 2
    return n
def hashFunction(key):
    capacity=getPrime(10)
    return key % capacity
def insertData(key,data):
    index = hashFunction(key)
```

```
    hashTable[index]=[key,data]

def removeData(key):
    index=hashFunction(key)
    hashTable[index]=0

insertData(123,"C")
insertData(432,"DBMS")
insertData(123,"JAVA")
insertData(123,"C++")
print(hashTable)
removeData(123)
print(hashTable)
```

OUTPUT:



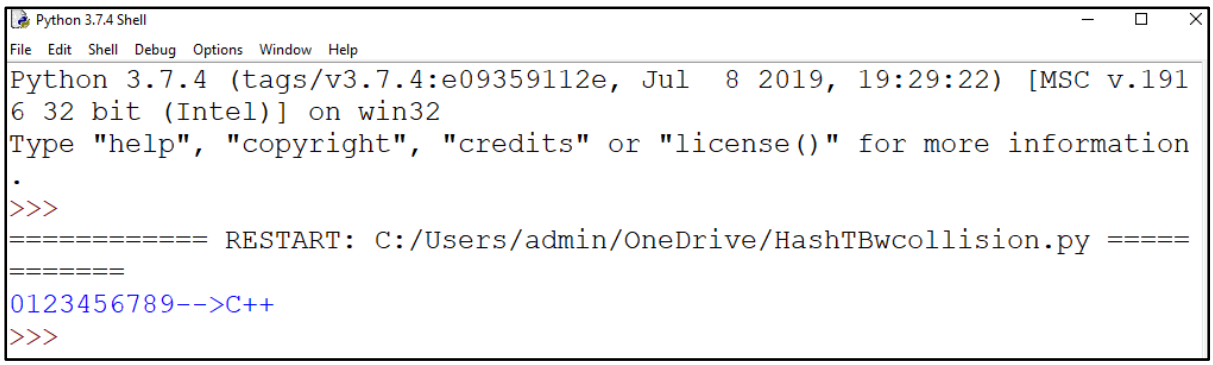
Practical 7: Write a program to create basic hash table for insertion, deletion, traversal, operation(Assume there is no collision)

(HashTBwcollision.py)

```
#Function to display hashtable
def display_hash(hashTable):
    for i in range(len(hashTable)):
        print(i,end="")

    for j in hashTable[i]:
        print("-->",end="")
        print(j,end="")
        print()

#creating Hashtable as a nested list
hashTable = [[] for _ in range(10)]
#Hashing Function to return key for every value
def Hashing(keyvalue):
    return keyvalue%len(HashTable)

#Insert function to add values to hash table
def insert(Hashtable,keyvalue,value):
    hash_key = Hashing(keyvalue)
    hashtable[hash_key].append(value)

#Driver code
```

```
insert(HashTable,11,'Java')
insert(HashTable,3,'Python')
insert(HashTable,10,'C')
insert(HashTable,9,'C++')
insert(HashTable,21,'DBMS')
insert(HashTable,20,'HTML')
insert(HashTable,4,'PHP')

display_hash(HashTable)
```

OUTPUT:



Practical 8: Write a program to implement priority queue with insertion, deletion, traversal operations

```
# Priority queue implementation in python

# function to heapify the tree
def heapify(arr,n,i):
    #find the largest among root, left child and right child
    largest = i
    l = 2*i+1
    r = 2*i+2
    if l < n and arr[i] < arr[l]:
        largest = l
    if r < n and arr[largest] < arr[r]:
        largest = r
    #swap and continue heapifying if root is not largest
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr,n,largest)
#function to insert an element into tree
def insert(array, newNum):
    size = len(array)
    if size == 0:
        array.append(newNum)
    else:
        array.append(newNum)
    for i in range((size//2)-1, -1, -1):
```

```
        heapify(array,size,i)
#function to delete an element from the tree
def deleteNode(array, num):
    size = len(array)
    i = 0
    for i in range(0, size):
        if num == array[i]:
            break
    array[i],array[size-1] = array[size-1],array[i]
    array.remove(size-1)
    for i in range((len(array)//2) -1, -1, -1):
        heapify(array, len(array), i)
arr = []
insert(arr,3)
insert(arr,4)
insert(arr,9)
insert(arr,5)
insert(arr,2)
print('Max-heap array:' + str(arr))
deleteNode(arr,4)
print('after deleting an element:' + str(arr))
```

OUTPUT: