1a Wapp to implement stack data structure

```
print("Moosa Ansari | 210 | SYIT")
# initial empty stack
my_stack = []

# append() function to push element in the my_stack
my_stack.append('x')
my_stack.append('y')
my_stack.append('z')

print(my_stack)

# pop() function to pop element from my_stack in LIFO order
print('\nElements poped from my_stack:')
print(my_stack.pop())
print(my_stack.pop())
print(my_stack.pop())

print('\nmy_stack after elements are popped:')
print(my_stack)
```

1b wapp to convert the entered infix expression into its postfix form

```
print("Moosa Ansari | 210 | SYIT")
OPERATORS = set(['+', '-', '*', '/', '(', ')', '^'])  #set of
operators

PRIORITY = {'+':1, '-':1, '*':2, '/':2, '^':3} # dictionary having
priorities

def infix_to_postfix(expression): #input expression
    stack = [] # initially stack empty
    output = '' # initially output empty

    for ch in expression:
        if ch not in OPERATORS:  # if an operand then put it
directly in postfix expression
            output+= ch
        elif ch=='(':  # else operators should be put in stack
            stack.append('(')
        elif ch==')':
            while stack and stack[-1]!= '(':
                output+=stack.pop()
            stack.pop()
        else:
            # lesser priority can't be on top on higher or equal
priority so pop and put in output
            while stack and stack[-1]!='(' and
PRIORITY[ch]<=PRIORITY[stack[-1]]:
                output+=stack.pop()
            stack.append(ch)
```

```
    while stack:
        output+=stack.pop()
    return output

expression = input('Enter infix expression: ')
print('infix expression: ',expression)
print('postfix expression: ',infix_to_postfix(expression))
```

```
2a Wapp to implement diff ops on array
print("Moosa Ansari | 210 | SYIT")
#FIRST EXECUTE WITH THIS CODE, THEN CHANGE array_adt.set(4, 9) TO
array_adt.set(5, 9) TO GET "INDEX ARRAY OUT OF BOUNDS" AND TAKE
NOTE OF BOTH OUTPUTS
class ArrayADT:
    def __init__ (self, capacity): #initialising array
         self.capacity=capacity
         self.array=[None]*capacity

    def get(self, index):
        if 0 <= index < self.capacity:
            return self.array[index]
        else:
            raise IndexError("Index Out Of Bounds")

    def set(self, index, value):
        if 0<= index < self.capacity:
                self.array[index] = value
        else:
              raise IndexError("Index out of bounds")

    def size(self):
            return self.capacity

# Example usage:
if __name__ == "__main__":
    array_adt = ArrayADT(5)  # Create an array ADT of capacity 5
    print("Operations On Array")

    print("Array capacity:", array_adt.size())

    # Set values in the array
    array_adt.set(0, 1)
    array_adt.set(1, 3)
    array_adt.set(2, 5)
    array_adt.set(3, 7)
    array_adt.set(4, 9)

    # Get values from the array and print
    print("\nArray elements:")
    for i in range(array_adt.size()):
```

```
        print(array_adt.get(i))
```

2.b.1 Wapp to perform multiplication of two matrices

```
print("Moosa Ansari | 210 | SYIT")
print("Multiplication of 2 Matrices")

#3x3 Matrix
X=[[1,2,3],
   [4,5,6],
   [7,8,9]]
#3x4 Matrix
Y= [[9,8,7,6],
    [5,4,3,2],
    [1,0,9,8]]
#Result is 3x4
result=[[0,0,0,0],
        [0,0,0,0],
        [0,0,0,0]]

#iterating through rows of X
for i in range(len(X)):
    #iterating through cols of Y
    for j in range(len(Y[0])):
        #iterating 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)
```

2b2 Wapp to perform matrix transpose

```
print("Moosa Ansari | 210 | SYIT")
print("Transpose of a Matrix")

matrix = [[1, 2],
                  [3, 4],
                  [5, 6]]
rmatrix=[[0, 0, 0],
                [0, 0, 0]]
for i in range(len(matrix)):
    for j in range(len(matrix[0])):
        rmatrix[j][i] = matrix[i][j]
for r in rmatrix:
    print(r)
```

2b3 WPP to perform addition of two matrix

```
print("Moosa Ansari | 210 | SYIT")
print("Addition of 2 Matrices")
#3x3 Matrix
X=[[1,2,3],
```

```
   [4,5,6],
   [7,8,9]]
#3x3 Matrix
Y= [[9,8,7,],
    [6,5,4,],
    [3,2,1]]
#Result is 3x4
result=[[0,0,0],
        [0,0,0],
        [0,0,0,]]

#iterating through rows of X
for i in range(len(X)):
    #iterating through cols of Y
    for j in range(len(Y[0])):
            result[i][j] = X[i][j]+Y[i][j]

for r in result:
    print(r)
```

2c wapp to merge two sorted arrays

```
print("Moosa Ansari | 210 | SYIT")
print("Merging Two Matrices")

def mergeArrays(arr1, arr2, n1, n2, arr3):
    i=0
    j=0
    k=0

    while(i < n1):
        arr3[k] = arr1[i]
        k += 1
        i += 1

    while(j < n2):
        arr3[k] = arr2[j]
        k += 1
        j += 1

    arr3.sort()

if __name__=='__main__':
    arr1=[1,3,5,7]
    n1=len(arr1)

    arr2=[2,4,6,8]
    n2=len(arr2)

    arr3 = [0 for i in range(n1+n2)]
    mergeArrays(arr1, arr2, n1, n2, arr3)
```

```
    print("Array after merging")
    for i in range(n1+n2):
        print(arr3[i], end=" ")
```

3a WAPP to implement queue

```
print("Moosa Ansari | 210 | SYIT");
from queue import Queue #Inbuild Function "Queue"
q=Queue(maxsize=4)#Defining the size of function "Queue"
print("Initial Size Before Insertion:",q.qsize())
q.put('A') #to put elements in the Queue
q.put('AA')
q.put('AAA')
q.put('AAAA')
print("After Insertion:",q.qsize()) #will check the size of queue
print("Queue is Full or Not:",q.full()) #will see if queue is full
or not
print("Size of Queue:",q.qsize()) #will check the size of queue
print("Removing Elements:")
print(q.get()) #to get elements from the Queue
print(q.get())
print("Empty or Not??",q.empty())
print("Again Removing Elements:")
print(q.get())
print(q.get())
print("Empty or Not??",q.empty())
print("Size of Queue:",q.qsize()) #will check the size of queue
```

3b Wapp to implement Dequeue operations

```
print("Moosa Ansari | 210 | SYIT");
#importing collections for deque operations
import collections
#initializing deque
de = collections.deque([1, 2, 3])
print("deque: ", de)

#using append() to insert element at rig end
#inserts 4 at the end of deque
de.append(4)

#printing modified deque
print("\nThe deque after appending at right is: ")
print(de)

#using appendleft() to insert element at left end
#inserts 6 at the beginning of deque
de.appendleft(6)

#printing modified deque
```

```
print("\nThe deque after appending at left is: ")
print(de)
```

3c Wapp to implement Circular Queue

```
print("Moosa Ansari | 210 | SYIT");
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 is 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 is 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())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
```

```
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
```

4aIterative Python program to search an element in linked list

```
print("Moosa Ansari | 210 | SYIT");
#Node class
class Node:

    #Function to initialise the
    #node object
    def __init__(self, data):

        # Assign data
        self.data = data

        # Initialize next as null
        self.next = None

#Linked List class
class LinkedList:
    def __init__(self):

        # Initialize head as None
        self.head = None

    # This function insert a new node at the
    # beginning of the linked list
    def push(self, new_data):

        # Create a new Node
        new_node = Node(new_data)

        #3. Make next of new Node as head
        new_node.next = self.head

        #4 Move the head to point to new Node
        self.head = new_node

    #This Function checks whether the value
    #x present in the linked list
    def search(self, x):

        #Initialize current to head
         current = self.head

        # Loop till current not equal to None
         while current != None:
            if current.data == x:
```

```
                # Data found
                  return True

            current = current.next

         return False

#Driver code
if __name__ == '__main__':

    # Start with the empty list
    llist = LinkedList()

    # Use push() to construct list
    #14->21->11->30->10
    llist.push(10);
    llist.push(30);
    llist.push(11);
    llist.push(21);
    llist.push(14);

    #print(llist)
    if llist.search(21):
        print("Yes")
    else:
        print("No")
```

4b Reverse

```
# Python program to reverse a linked list
# Time Complexity : O(n)
# Space Complexity : O(1)
print("Moosa Ansari | 210 | SYIT");
# Node class
class Node:

    # Constructor to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    # Function to reverse the linked list
    def reverse(self):
        prev = None
        current = self.head
        while(current is not None):
```

```
            next = current.next
            current.next = prev
            prev = current
            current = next
        self.head = prev

    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    # Utility function to print the LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print(temp.data, end=" ")
            temp = temp.next


# Driver code
llist = LinkedList()
llist.push(40)
llist.push(30)
llist.push(20)
llist.push(10)

print ("Given linked list")
llist.printList()
llist.reverse()
print ("\nReversed linked list")
llist.printList()


#5a WAPP To implement Linear Search
print("Moosa Ansari | 210 | SYIT")
def linear_Search(list1, n, key):

    # Searching list1 sequentially
    for i in range(0, n):
        if (list1[i] == key):
            return i
    return -1

list1 = [1,3, 5, 4, 7, 9]
key = 7

n = len(list1)
res = linear_Search(list1, n, key)
if(res == -1):
    print("Element not found")
else:
    print("Element found at index: ", res)
```

```
#5b WAPP To implement Binary Search
print("Moosa Ansari | 210 | SYIT")
def binary_search(list1,n):
    low = 0
    high = len(list1)-1
    mid = 0

    while low<=high:
        mid=(high + low)//2

        if list1[mid] < n:
            low = mid + 1

        elif list1[mid]>n:
            high = mid - 1

        else:
            return mid

    return -1

list1 = [12, 24, 32, 39, 45, 50, 54]
n=50

result = binary_search(list1, n)

if result != -1:
    print("Element is present at index", str(result))
else:
    print("Element not present in list1")
```

```
#6a WAPP To implement Bubble Sort
print("Moosa Ansari | 210 | SYIT")
def bubble_sort(list1):
    for i in range(0,len(list1)-1):
        for j in range(len(list1)-1):
            if(list1[j]>list1[j+1]):
                temp = list1[j]
                list1[j] = list1[j+1]
                list1[j+1] = temp
    return list1

list1 = [5, 3, 8, 6, 7, 2]
print("The unsorted list is: ",list1)
print("The sorted list is: ",bubble_sort(list1))
```

```
#6b WAPP to Implement Selection Sort
```

```
print("Moosa Ansari | 210 | SYIT")
def selection_sort(array):
    length = len(array)

    for i in range(length-1):
        minIndex = i

        for j in range(i+1, length):
            if array[j] < array[minIndex]:
                minIndex = j

        array[i], array[minIndex] = array[minIndex], array[i]

    return array
array = [21,6,9,33,3]

print("The unsorted array is: ",array)
print("The sorted array is: ",selection_sort(array))



#6c WAPP to Implement Insertion Sort (HW)



#7a Binary tree
print("Moosa Ansari | 210 | SYIT")
class Node:
    def _init_(self, data):
        self.left = None
        self.right = None
        self.data = data

    #Compare the new value with the parent node
    def insert(self, data):
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                    if self.right is None:
                        self.right = Node(data)
                    else:
                        self.right.insert(data)
        else:
            self.data = data

    #Print The Tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.data)
```

```
        if self.right:
            self.right.PrintTree()

#Use the insert method to add nodes
root = Node(12)
root.insert(6)
root.insert(14)
root.insert(3)
root.PrintTree()



#7b WAPP To demonstrate Tree Traversing (In Order, PostOrder,
PreOrder Traversal)

print("Moosa Ansari | 210 | SYIT")
class Node:
    def _init_(self, key):
            self.left = None
            self.right = None
            self.val = key

#A function to do inorder tree traversal
def printInorder(root):
    if root:

        # First recur on left child
        printInorder(root.left)

        #then print the data of node
        print(root.val)

        #Now recur on right child
        printInorder(root.right)

#A function to do postorder tree traversal
def printPostorder(root):
    if root:

        # First recur on left child
        printPostorder(root.left)

        # then recur on right child
        printPostorder(root.right)

        #Now print the data of node
        print(root.val)

#A function to do PreOrder tree traversal
def printPreorder(root):
     if root:

         # First print the data of the node
         print(root.val)
```

```
        # Then recur on left child
        printPreorder(root.left)

        # Finally recur on right child
        printPreorder(root.right)

#Driver Code
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)



#8aHeapDeletion
print("Moosa Ansari | 210 | SYIT")
import heapq

# Sample heap
heap = [10, 20, 15, 25, 30, 40, 50]

#Convert the list into a heap
heapq.heapify(heap)
print("Heap before deletion:", heap)

#Deleting an element from the heap
element_to_delete = 15

heap.remove(element_to_delete)
heapq.heapify(heap)

# Rebuild the heap after removal
print(f"Deleted element {element_to_delete}.\nHeap after
deletion:",heap)



#8b HeapInsertion
print("Moosa Ansari | 210 | SYIT")
import heapq

# Sample heap
heap = [10, 20, 15, 25, 30, 40, 50]
```

```
#Convert the list into a heap
heapq.heapify(heap)
print("Heap before insertion:", heap)

#Deleting an element from the heap
element_to_insert = 60
heapq.heappush(heap, element_to_insert)

# Rebuild the heap after removal
print(f"Inserted element {element_to_insert}.\nHeap after
insertion:",heap)



#9a Graph Traversal
print("Moosa Ansari | 210 | SYIT")
#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_edges(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 dictionary
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_edges(graph))



#9b Using a Python dictionary to act as an adjacency list
print("Moosa Ansari | 210 | SYIT")
graph={
    '5' : ['3','7'],
    '3' : ['2', '4'],
    '7' : ['8'],
    '2' : [],
    '4' : ['8'],
    '8' : []
}

visited = set() # Set to keep track of visited nodes of the graph.

def dfs(visited, graph, node): # Function for DFS
    if node not in visited:
        print(node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

#Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')



#9c BFS algorithm in Python
print("Moosa Ansari | 210 | SYIT")
import collections

#BFS algorithm
def bfs(graph, root):

    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:

        # Dequeue a vertex from queue
        vertex = queue.popleft()
        print(str(vertex) +" ", end="")

        # If not visited, mark it as visited, and enqueue it
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)

if __name__ == '__main__':
    graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
```

```
print("Following is Breadth First Traversal:")
bfs(graph, 0)
```