

# LAB ASSIGNMENT-11.1

Name: S.varun

Roll.No:2403A53L02

Batch-46

## Task-1

Generate a Python Stack class with push, pop, peek and is\_empty methods.

Include docstrings and example usage.

```
class Stack:    def  
    __init__(self):  
        self.items = []  
  
    def push(self, item):  
        self.items.append(item)  
  
    def pop(self):      if not  
        self.is_empty():  
            return self.items.pop()    def  
  
    peek(self):       if not  
        self.is_empty():  
            return self.items[-1]    def  
  
    is_empty(self):  
        return len(self.items) == 0  
  
s = Stack()  
s.push(10)
```

```
s.push(20)
print(s.peek())
print(s.pop())
```

## Task-2

Create a FIFO Queue class using Python lists with enqueue, dequeue, peek and size methods. Include simple testing code.

```
class Queue:
    def __init__(self):
        self.items = []
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        if self.items:
            return self.items.pop(0)
    def peek(self):
        if self.items:
            return self.items[0]
    def size(self):
        return len(self.items)
q = Queue()
q.enqueue(1)
q.enqueue(2)
print(q.dequeue())
```

## Task-3

Generate a Singly Linked List in Python with Node class, insert method and display method.

```

class Node:    def
    __init__(self, data):
        self.data = data
    self.next = None class
LinkedList:   def
    __init__(self):
        self.head = None    def
    insert(self, data):      new
        = Node(data)
        if not self.head:
            self.head = new
            return
        temp  =  self.head
        while      temp.next:
            temp  =  temp.next
        temp.next = new    def
        display(self):      temp
        = self.head      while
        temp:
            print(temp.data, end=" ->
")
            temp = temp.next
print("None") l = LinkedList()
l.insert(5)
l.insert(10)
l.display()

```

## Task-4

Create a Binary Search Tree class with recursive insert and inorder traversal methods in Python.

```

class Node:
    def __init__(self, key):
        self.key = key
        self.le = None
        self.right = None
    class BST:
        def insert(self, root, key):
            if root is None:
                return Node(key)
            if key < root.key:
                root.le = self.insert(root.le, key)
            else:
                root.right = self.insert(root.right, key)
            return root
        def inorder(self, root):
            if root:
                self.inorder(root.le)
                print(root.key, end=" ")
                self.inorder(root.right)
    tree = BST()
    root = None
    for i in [50, 30, 70, 20]:
        root = tree.insert(root, i)
    tree.inorder(root)

```

## Task-5

Implement a HashTable class using chaining with insert, search and delete operations.

```
class HashTable:  
    def __init__(self, size=10):  
        self.size = size      self.table =  
        [[] for _ in range(size)]      def  
        hash(self, key):  
            return key % self.size  
        def insert(self, key):  
            index = self.hash(key)  
            self.table[index].append(key)  
        def search(self, key):      index  
            = self.hash(key)      return key  
            in self.table[index]      def  
            delete(self, key):      index =  
            self.hash(key)      if key in  
            self.table[index]:  
                self.table[index].remove(key)  
        h = HashTable()  
        h.insert(15)  
        print(h.search(15))
```

## Task-6

Generate a Graph class using adjacency list with methods add\_vertex, add\_edge and display.

```

class Graph:    def
    __init__(self):
        self.graph = {}
    def add_vertex(self, v):
        if v not in self.graph:
            self.graph[v] = []
    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)
    def display(self):
        for v in self.graph:
            print(v, ":", self.graph[v])
g = Graph()
g.add_vertex("A")
g.add_vertex("B")
g.add_edge("A", "B")
g.display()

```

## Task-7

Create a PriorityQueue class using heapq module with enqueue(priority), dequeue and display methods.

```

import heapq
PriorityQueue:
    def __init__(self):
        self.heap = []
    def enqueue(self, item):

```

```

heapq.heappush(self.heap, item)

def dequeue(self):
    if self.heap:      return
    heapq.heappop(self.heap)  def
display(self):      print(self.heap)
p = PriorityQueue()
p.enqueue(3)
p.enqueue(1) print(p.dequeue())

```

## Task-8

Implement a double-ended queue using collections.deque with insert/remove from both ends.

```

from collections import deque

class DequeDS:
    def __init__(self):
        self.d = deque()  def
    insert_front(self, x):
        self.d.appendleft(x)  def
    insert_rear(self, x):
        self.d.append(x)
    def remove_front(self):
        return self.d.popleft()  def
    remove_rear(self):
        return self.d.pop()
dq = DequeDS()
dq.insert_front(5)

```

```
dq.insert_rear(10)  
print(dq.remove_front())
```

## Task-9

Design a Campus Resource Management System by choosing suitable data structures (Stack, Queue, Priority Queue, Linked List, BST, Graph, Hash Table, Deque) for attendance tracking, event registration, library borrowing, bus scheduling, and cafeteria queue, give a feature→structure→justification and implement one feature in Python with comments.

```
class CafeteriaQueue:  
    def __init__(self):  
        self.queue = []  
    def arrive(self, student):  
        self.queue.append(student)  
        print(student, "joined the queue")  
    def serve(self):  
        if self.queue:  
            served = self.queue.pop(0)  
            print(served, "served")  
        else:  
            print("No students in queue")  
    def display(self):  
        print("Current  
Queue:", self.queue)  
cq = CafeteriaQueue()  
cq.arrive("Akhil")  
cq.arrive("Rahul")  
cq.display()  
cq.serve()  
cq.display()
```

## Task-10

Create a Smart E-Commerce Platform design by selecting appropriate data structures (Stack, Queue, Priority Queue, Linked List, BST, Graph, Hash Table, Deque) for shopping cart, order processing, top-selling tracker, product search, and delivery routes, include a feature→structure→justification table and implement one feature as a clean Python program with docstrings.

```
class OrderQueue:  
    def __init__(self):  
        self.orders = []  
    def place_order(self, order):  
        self.orders.append(order)  
        print(order, "placed")  
    def process_order(self):  
        if self.orders:  
            processed = self.orders.pop(0)  
            print(processed, "processed")  
        else:  
            print("No orders")  
    def show_orders(self):  
        print("Pending Orders:", self.orders)  
  
oq = OrderQueue()  
oq.place_order("Order101")  
oq.place_order("Order102")  
oq.show_orders()  
oq.process_order()  
oq.show_orders()
```