Usama Mehboob **Roll No 225** **Section E**

LAB # 10

LIST, TUPLE, DICTIONARY, CLASS OBJECT AND IMPLEMENT PRIORITY QUEUE

OBJECTIVE

Familiarization with python language using list, tuple, dictionary, class and object and Manage a set of records with priority queue using queue and heapq module in python.

Lab Tasks:

1. Store the names of a few of your friends in a list called names. Print each person's name by accessing each element in the list, one at a time.

Code:

```
# List of friends' names
names = ['Rohan', 'Murtaza', 'Shayan', 'Usama', 'Fasahat']

# Print each name one by one
for name in names:
    print(name)
```

Output:

```
Rohan
Murtaza
Shayan
Usama
Fasahat
```

2. If you could invite anyone, living or deceased, to dinner, who would you invite? Make a list that includes at least three people you'd like to invite to dinner. Then use your list to print a message to each person, inviting them to dinner.

Code:

```
guests = ['Emaad', 'Shayan', 'Dua']

for guest in guests:
    print(f"Dear {guest},\nI would be honored to have you join me for dinner.\nBest regards,\nYour Host\n")
```

Output:

```
Dear Emaad,
I would be honored to have you join me for dinner.
Best regards,
Your Host

Dear Shayan,
I would be honored to have you join me for dinner.
Best regards,
Your Host

Dear Dua,
I would be honored to have you join me for dinner.
Best regards,
Your Host
```

3. Changing Guest List: You just heard that one of your guests can't make the dinner, so you need to send out a new set of invitations. You'll have to think of someone else to invite.

- Modify your list, replacing the name of the guest who can't make it with the name of the new person you are inviting.
- Print a second set of invitation messages, one for each person who is still in your list.

Code:

```
guests = ['Emaad', 'Shayan', 'Dua']

guests[2] = 'Saqib'
for guest in guests:
    print(f"Dear {guest},\nI would be honored to have you join me for dinner.\nBest regards,\nYour Host\n")
```

Output:

```
Dear Emaad,
I would be honored to have you join me for dinner.
Best regards,
Your Host

Dear Shayan,
I would be honored to have you join me for dinner.
Best regards,
Your Host

Dear Saqib,
I would be honored to have you join me for dinner.
Best regards,
Your Host
```

4. Create a Class “Employee”, it’s a common base class for all the employee. Then initialize employee’s parameter like empName and salary and create function like displayCount() contain total number of employee in your knowledge base and displayEmployee() contain empName and their salary.

Code:

```
class Employee:
    employee_count = 0

    def __init__(self, empName, salary):

        self.empName = empName
        self.salary = salary

        Employee.employee_count += 1

    @staticmethod
    def displayCount():
        print(f"Total number of employees: {Employee.employee_count}")

    def displayEmployee(self):
        print(f"Employee Name: {self.empName}, Salary: {self.salary}")

emp1 = Employee('Usama', 50000)
emp2 = Employee('Shayan', 60000)
emp3 = Employee('Rohan', 55000)

Employee.displayCount()

emp1.displayEmployee()
emp2.displayEmployee()
emp3.displayEmployee()
```

Output:

```
Total number of employees: 3
Employee Name: Usama, Salary: 50000
Employee Name: Shayan, Salary: 60000
Employee Name: Rohan, Salary: 55000
```

5.In contrast to the standard FIFO implementation of Queue, the LifoQueue uses last-in, first- out ordering (normally associated with a stack data structure). Implement LIFO queue using queue module.

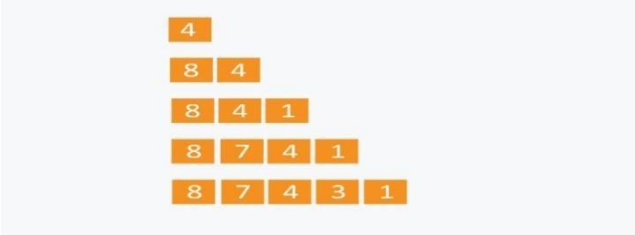
Code:

```
import queue

lifo_queue = queue.LifoQueue()

lifo_queue.put(10)
lifo_queue.put(20)
lifo_queue.put(30)

print("Items dequeued (pop from LIFO queue):")
while not lifo_queue.empty():
    item = lifo_queue.get()
    print(item)
```

Output:

```
Items dequeued (pop from LIFO queue):
30
20
10
```

6. We have an array of 5 elements: [4, 8, 1, 7, 3] and we have to insert all the elements in the max-priority queue. First as the priority queue is empty, so 4 will be inserted initially. Now when 8 will be inserted it will move to front as 8 is greater than 4. While inserting 1, as it is the current minimum element in the priority queue, it will remain in the back of priority queue. Now 7 will be inserted between 8 and 4 as 7 is smaller than 8. Now 3 will be inserted before 1 as it is the 2nd minimum element in the priority queue. All the steps are represented in the diagram below:



Usama Mehboob **Roll No 225** **Section E**

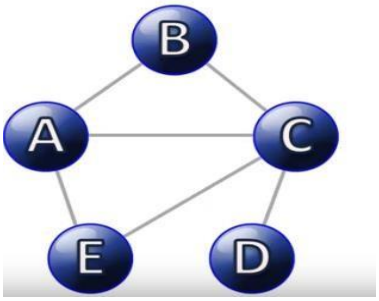
Code:

```
import heapq
def max_priority_queue(arr):
    pq = []

    for num in arr:

        heapq.heappush(pq, -num)
        print(f"Inserted {num}: {[-x for x in pq]}")


    print("\nPriority Queue after all insertions (max-priority order):")
    while pq:
        max_value = -heapq.heappop(pq)
        print(max_value)
arr = [4, 8, 1, 7, 3]
max_priority_queue(arr)
```

Output:

```
Inserted 4: [4]
Inserted 8: [8, 4]
Inserted 1: [8, 4, 1]
Inserted 7: [8, 7, 1, 4]
Inserted 3: [8, 7, 1, 4, 3]

Priority Queue after all insertions (max-priority order):
8
7
4
3
1
```

7. Implement graph using adjacency list using list or dictionary, make a class such as Vertex and Graph then make some function such as add_nodes, add_edges, add_neighbors, add_vertex, add_vertices and suppose whatever you want to need it.




```
class Vertex:
    def __init__(self, value):
        self.value = value
        self.neighbors = []  # List to hold neighbors

    def add_neighbor(self, vertex):
        if vertex not in self.neighbors:
            self.neighbors.append(vertex)


class Graph:
    def __init__(self):
        self.vertices = {}  # Dictionary to hold vertices

    def add_vertex(self, value):
        if value not in self.vertices:
            self.vertices[value] = Vertex(value)

    def add_vertices(self, values):
        for value in values:
            self.add_vertex(value)

    def add_edge(self, value1, value2):
        if value1 in self.vertices and value2 in self.vertices:
            self.vertices[value1].add_neighbor(value2)
            self.vertices[value2].add_neighbor(value1)  # For undirected graphs
        else:
            print(f"One or both vertices '{value1}' or '{value2}' are not in the graph.")

    def display_graph(self):
        for key in self.vertices:
            neighbors = ", ".join(self.vertices[key].neighbors)
            print(f"{key}: {neighbors}")
```

```
# Add edges based on the image
graph.add_edge('A', 'B')
graph.add_edge('A', 'E')
graph.add_edge('A', 'C')
graph.add_edge('B', 'C')
graph.add_edge('C', 'D')

graph.display_graph()
```

Output:

```
A: B, E, C
B: A, C
C: A, B, D
D: C
E: A
```