Experiment No : 5

Dated : 14th Feb 2022

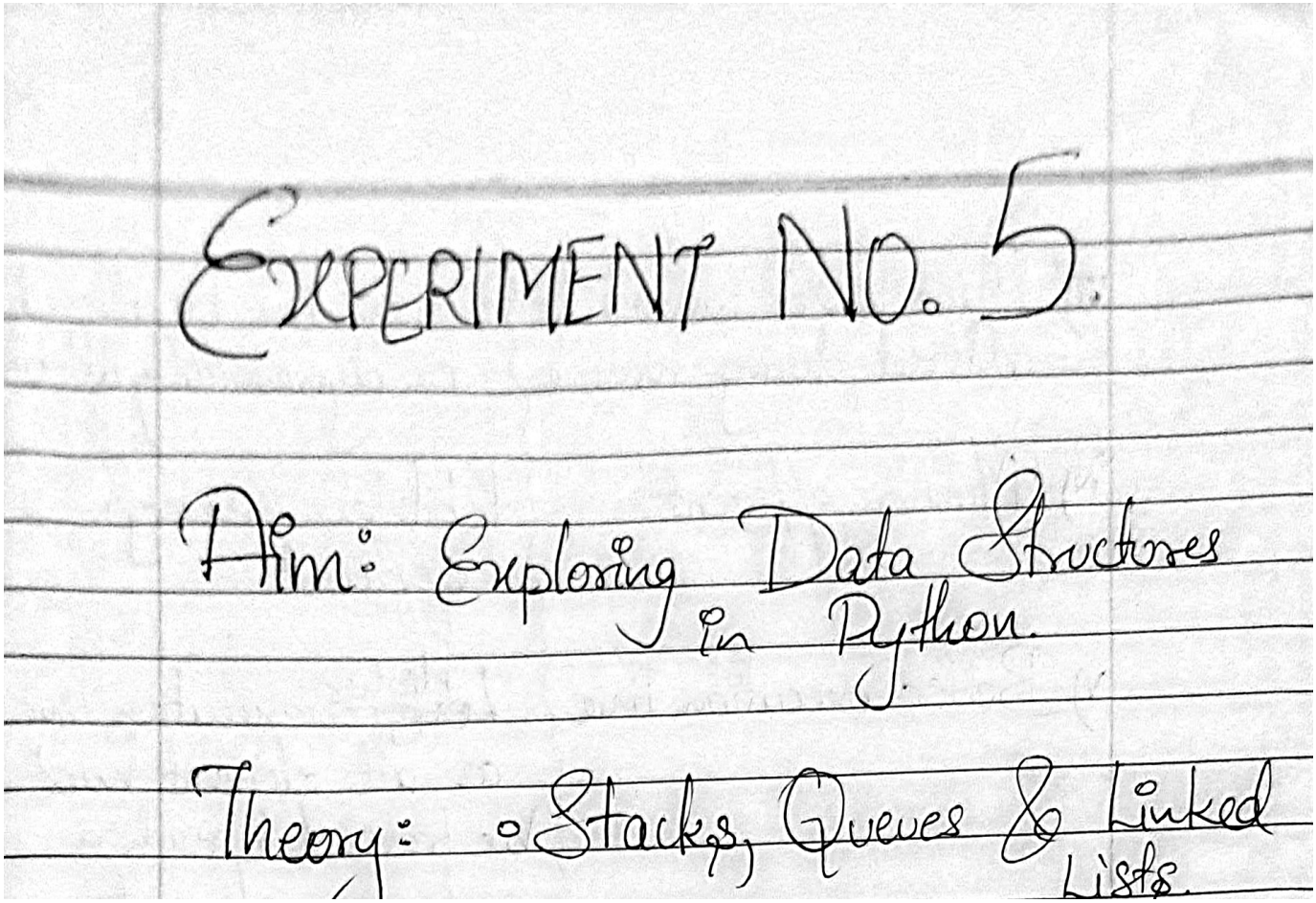
Aim

Exploring Data Structures in Python

Theory

- Stacks, Queues, Deques & Linked List
  - Difference between Stacks & Queues (enlist at least 3)
  - Difference between Arrays & Linked List (enlist at least 5)
  - Difference between Queue & Deque (enlist at least 3)

Handwritten Theory :



* Difference btw Stacks & Queues.

i) Stack follows the LIFO principle	Queues follow the FIFO principle.
ii) The insert operation is called push	Insert operation is called append/enqueue.
iii) Delete operation is called pop	Delete operation is called dequeue

* Difference btw Arrays & Linked Lists.

i) Arrays is a collection of elements of similar data type.	A linked list is a collection of objects known as nodes
ii) Array elements are stored in contiguous memory locations	Linked list objects/nodes can be stored anywhere in the memory

iii) Thier sizes cannot be altered during runtime	Sizes can be altered or dynamically allotted
iv) Memory efficient	A bit less memory efficient
v) Better execution time	~~Lower~~ Higher execution time as all elements must be traversed to reach a particular element

* Difference btw ~~E~~queue & Dequeue

i) A classic Queue can insert elements from the rear & delete from the front.	Depending on its type a deque can either insert or delete elements from both sides
ii) Hence, a queue follows FIFO principle.	Deque may or may not follow FIFO principle.
iii) More memory efficient but less versatile.	Less memory efficient but more versatile with multiple real life uses.

Programs to be performed : (Attempt any 1)

1.

Create a python package called ‘stack_app’ where the following modules are implemented : (at least 2 modules) and demonstrate their usage.

- in_post.py (for Infix to Postfix conversion)
- post_eval.py (for Postfix Evaluation)
- in_pre.py (for Infix to Prefix conversion)
- pre_post.py (to convert prefix to postfix)

2.

Create a python package called ‘linkedlist_ops’ where the following modules are implemented: (at least 2 modules) and demonstrate their usage.

- del_dup.py (to remove the duplicate elements in the LL)
- sort_ll.py (to sort the elements in a LL)
- merge_ll.py (to merge the two given LLs after sorting)
- add_ll.py (to sum all the elements in the LL)

3.

3. Write an implementation of the Priority Queue ADT using List. Also keep the list sorted so that removal is a constant time operation.

```
class PriorityQueue(list):

  def enQueue(self,data, priority):
    if(len(self)==0):
      self.append([data,priority])
      return
    i = len(self) - 1
    while(i>=0 and priority<self[i][1]):
        i = i-1
    self.insert(i+1,[data,priority])

  def deQueue(self):
    self.pop(0)
```

```
  def displayQueue(self):
    print('\nThe current queue is: ')
    print("Front")
    for ele in self:
      print(ele[0])
    print("Rear")

  def displayPriority(self):
    for i in self:
      print("Priority:",i[0],"Value:",i[1])
```

```
p = PriorityQueue()
p.enQueue(3,2)
p.enQueue(13,1)
p.enQueue(9,2)
p.displayQueue()
```

```
The current queue is:
Front
13
3
9
Rear
```

```
p.enQueue(1,2)
p.enQueue(5,1)
p.displayPriority()
```

```
Priority: 13 Value: 1
Priority: 5 Value: 1
Priority: 3 Value: 2
Priority: 9 Value: 2
Priority: 1 Value: 2
```

```
p.deQueue()
p.deQueue()
p.displayQueue()
```

```
The current queue is:
Front
3
9
1
Rear
```