Name of student: Abhay Omprakash Prajapati		
Roll no: 41		**Tutorial No:** **Assignment : 1**
Title of LAB Assignment: : Data Structure & Threads in Python		
DOP: 31-10-2023		**DOS:02-10-2023**
CO Mapped:	**PO Mapped:**	**Signature:**

Q1. Explain multithreading in Python:

Multithreading in Python refers to the ability of a program to run multiple threads concurrently. A thread is a lightweight sub-process that can perform tasks independently of the main program. This allows the program to execute multiple operations simultaneously, which is particularly useful for tasks that can be parallelized.

Python provides the threading module for managing threads. It allows you to create, start, and control threads. Each thread runs in its own space, allowing it to execute separate tasks concurrently. However, it's important to be mindful of thread safety and synchronization to prevent conflicts when multiple threads access shared resources.

Q2. Define the two types of data structures and give differences between them:

The two types of data structures are:

1. **Linear Data Structures:**

In linear data structures, elements are stored sequentially, and each element is connected to its previous and next element.
Examples include arrays, linked lists, stacks, and queues.
They are suitable for situations where data needs to be organized in a linear fashion.

2. **Non-Linear Data Structures:**

Non-linear data structures do not have a sequential arrangement of elements.
Examples include trees, graphs, heaps.
They are useful for representing relationships and hierarchies between elements.

Differences:

- Linear data structures store elements in a sequential manner, while non-linear data structures do not follow a specific order.
- Linear data structures are easier to implement and manipulate, while non-linear data structures are more complex and specialized.
- Accessing elements in linear data structures is generally faster, while operations in non-linear data structures can be more time-consuming.
- Linear data structures are suitable for tasks like searching, sorting, and accessing elements sequentially. Non-linear data structures are used for more complex relationships and hierarchies.

Q3. Explain the concept of stack and Queue:

Stack:

- A stack is a linear data structure where elements are added and removed from the same end, called the top.

- It follows the Last In, First Out (LIFO) principle, meaning the last element added to the stack is the first one to be removed.
- Common operations on a stack include push (add an element), pop (remove an element), and peek (retrieve the top element without removing it).
- Examples of stack applications include function call management in programming and undo functionality in applications.

Queue:

- A queue is a linear data structure where elements are added at the rear end and removed from the front end.
- It follows the First In, First Out (FIFO) principle, meaning the first element added to the queue is the first one to be removed.
- Common operations on a queue include enqueue (add an element), dequeue (remove an element), and peek (retrieve the front element without removing it).
- Examples of queue applications include scheduling tasks, breadth-first search in graph algorithms, and managing requests in computer systems.

Q4. Write an algorithm to implement insertion operation of the queue:

```python
class Queue:
   def __init__(self):
       self.items = []

   def is_empty(self):
       return self.items == []

   def enqueue(self, item):
       self.items.append(item)

   def dequeue(self):
       if not self.is_empty():
           return self.items.pop(0)
       else:
           return "Queue is empty"

   def peek(self):
       if not self.is_empty():
           return self.items[0]
       else:
           return "Queue is empty"

# Algorithm for Insertion Operation
def insert_into_queue(queue, item):
    queue.enqueue(item)

# Example Usage
q = Queue()
insert_into_queue(q, 10)
insert_into_queue(q, 20)
insert_into_queue(q, 30)
print("Queue:", q.items)
```

This algorithm implements the insertion operation of a queue using a class-based approach. The Queue class has methods to perform enqueue, dequeue, and peek operations. The insert_into_queue function inserts an item into the queue using the enqueue method. In the example usage, we create a queue, insert three items, and print the queue. The output will be **Queue: [10, 20, 30].**