Name of student: Abhay Omprakash Prajapati		
Roll no: 41		**Tutorial No: 6**
Title of LAB Assignment:To implement programs on Data Structures using Python		
DOP: 25-09-2023		**DOS:02-10-2023**
CO Mapped: Co1,Co2	**PO Mapped:** PO3 ,PO6	**Signature:**

1. Create, Traverse, Insert, and Remove Data Using Linked List

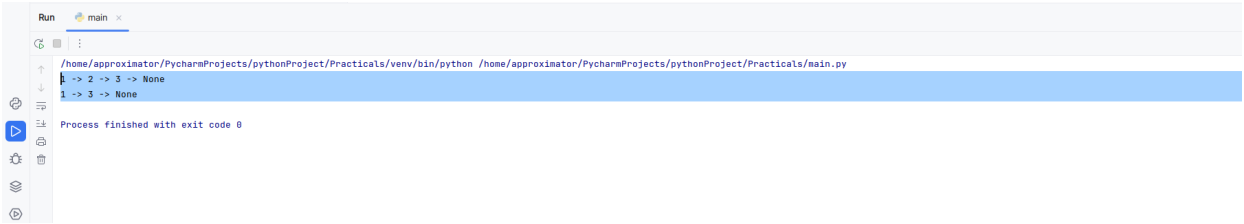
Aim: To create, traverse, insert, and remove data using a linked list.
Theory: In this task, we'll create a basic linked list structure and implement operations like insertion, traversal, and removal of data.
Code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
```

```
    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def remove(self, data):
        if self.head and self.head.data == data:
            self.head = self.head.next
            return
        current = self.head
        while current and current.next:
            if current.next.data == data:
                current.next = current.next.next
            current = current.next

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

linked_list = LinkedList()
linked_list.append(1)
linked_list.append(2)
linked_list.append(3)
linked_list.display()
linked_list.remove(2)
linked_list.display()
```

Conclusion: We successfully created a linked list, inserted data, and removed data from it.

Output:



2. Implementation of Stacks

Aim: To implement a stack data structure.
Theory: In this task, we'll create a basic stack structure and implement operations like push and pop.
Code:

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

    def is_empty(self):
        return len(self.items) == 0

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

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

stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
print("Popped:", stack.pop())
print("Popped:", stack.pop())
```

Conclusion: We successfully implemented a stack and demonstrated push and pop operations.
Output:

3. Implementation of Queue

Aim: To implement a queue data structure.
Theory: In this task, we'll create a basic queue structure and implement operations like enqueue and dequeue.
Code:

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

    def is_empty(self):
        return len(self.items) == 0

    def enqueue(self, item):
        self.items.insert(0, item)

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

queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print("Dequeued:", queue.dequeue())
print("Dequeued:", queue.dequeue())
```

Conclusion: We successfully implemented a queue and demonstrated enqueue and dequeue operations.
Output: The output will display the items dequeued from the queue.

4. Implementation of Dequeue (Double-Ended Queue)

Aim: To implement a double-ended queue (deque) data structure.
Theory: In this task, we'll create a basic deque structure and implement operations for both ends, such as inserting and removing elements.
Code:

```
from collections import deque

dq = deque()
dq.append(1)
dq.append(2)
dq.appendleft(3)
dq.appendleft(4)
```

```
print("Deque:", dq)

popped_from_right = dq.pop()
popped_from_left = dq.popleft()
print("Popped from right:", popped_from_right)
print("Popped from left:", popped_from_left)
```

Conclusion: We successfully implemented a deque and demonstrated append, appendleft, pop, and popleft operations.
Output: