Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL405**	**Course Name:**	**Skills Based Python Programming Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**5**
Title of the Experiment:	**To implement menu driven programs for Link List, Stack and Queue in python.**
Date of Performance:	**11/02/2025**
Date of Submission:	**18/02/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mr. Raunak Joshi

Signature :

Date:

Aim: To implement menu driven programs for Link List, Stack and Queue in python.

Theory:

Linked List Implementation in Python

Creation of Linked list

A linked list is created by using the node class.

We create a Node object and create another class to use this Node object. We pass the appropriate values through the node object to point to the next data elements.

Stack Implementation in Python:

A **stack** is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.

The functions associated with stack are:

empty() – Returns whether the stack is empty – Time Complexity: O(1) **size() –** Returns the size of the stack – Time Complexity: O(1)

top() / peek() – Returns a reference to the topmost element of the stack– Time Complexity: O(1)

push(a) – Inserts the element 'a' at the top of the stack – Time Complexity: O(1) **pop() –** Deletes the topmost element of the stack – Time Complexity: O(1)

Ways to implement Stack :

list

Collections.deque

queue.LifoQueue

Queue Implementation in Python:

Queue is a linear data structure that stores items in First In First Out (FIFO) manner.

A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition Time Complexity : O(1)

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition – Time Complexity : O(1)

Front: Get the front item from queue – Time Complexity : O(1)

Rear: Get the last item from queue – Time Complexity : O(1)

Queue in Python can be implemented by the following ways:

1. list
   collections
2. deque
3. queue.Queue

Write a Code for Implementing Linked list, Stack , Queue in python using any one of the appropriate method

Implementation:

```
stack = list ()

# Append Operation
stack.append ('a')
stack.append ('b')
stack.append ('c')
print ('Initial Stack')
print (stack)
```

```
Initial Stack
['a', 'b', 'c']
```

In [ ]:

```
# Pop Operation
print (stack.pop ())
print (stack.pop ())
print (stack.pop ())
print (stack)
```

```
c
b
a
[]
```

Given a valid parentheses string stringInput, return the nesting depth of stringInput. The nesting depth is the maximum number of nested parentheses.

Example 1: Input: s = "(1+(2*3)+((8)/4))+1" Output: 3 Explanation: Digit 8 is inside of 3 nested parentheses in the string.

Example 2: Input: s = "(1)+((2))+(((3)))" Output: 3 Explanation: Digit 3 is inside of 3 nested parentheses in the string.

Example 3: Input: s = "()(())((()()))" Output: 3

In [ ]:

```
class StackDepth:
    def maximumDepth(self, stringInput: str) -> int:
        max_depth = 0
        current_depth = 0

        for char in stringInput:
            if char == '(':
                current_depth += 1
                max_depth = max(max_depth, current_depth)
            elif char == ')':
                current_depth -= 1
```

```
        return max_depth


if __name__ == '__main__':
    stringInput = str(input("Enter the string: "))
    stack_depth = StackDepth()
    result = stack_depth.maximumDepth(stringInput)
    print("Output:", result)
```

```
Enter the string: (1+(2*3)+((8)/4))+1
Output: 3
```

Conclusion:

Comparison of stack using Queue module, and Queue using queue module

The queue module in Python is naturally designed for implementing queues using FIFO (First-In-First-Out) order. It provides efficient and thread-safe methods like put() and get() that make queue operations straightforward. However, implementing a stack (LIFO – Last-In-First-Out) using the same module is less efficient and requires additional logic, such as using two queues or reordering elements during insertion. This increases complexity and impacts performance. Therefore, while queues are best implemented using the queue module, stacks are better handled using Python's built-in list or collections.deque, which provide more intuitive and faster LIFO operations.