Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Ramos, Jan Lawrence M. | Engr. Maria Rizette H. Sayo |

October 11, 2025

# **Objectives**

Introduction

Another fundamental data structure is the queue. It is a close “the same” of the stack, as a queue is a collection of objects that are inserted and removed according to the first-in, first-out (FIFO) principle. That is, elements can be inserted at any time, but only the element that has been in the queue the longest can be next removed.

The Queue Abstract Data Type

Formally, the queue abstract data type defines a collection that keeps objects in a sequence, where element access and deletion are restricted to the first element in the queue, and element insertion is restricted to the back of the sequence. This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in, first-out (FIFO) principle. The queue abstract data type (ADT) supports the following two fundamental methods for a queue Q:

Q.enqueue(e): Add element e to the back of queue Q.

Q.dequeue( ): Remove and return the first element from queue Q;

an error occurs if the queue is empty.

The queue ADT also includes the following supporting methods (with first being analogous to the stack’s top method):

Q.first(): Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.

Q.is empty( ): Return True if queue Q does not contain any elements.

len(Q): Return the number of elements in queue Q; in Python, we implement this with the special method len .

This laboratory activity aims to implement the principles and techniques in:

* Writing Python program using Queues

Writing a Python program that will implement Queues operations

# **Methods**

Instruction: Type the python codes below in your Colab. Reconstruct them by implementing Queues (FIFO) algorithm. Hint: You may use Array or Linked List

# Stack implementation in python

# Creating a stack

def create\_stack():

    stack = []

    return stack

# Creating an empty stack

def is\_empty(stack):

    return len(stack) == 0

# Adding items into the stack

def push(stack, item):

    stack.append(item)

    print("Pushed Element: " + item)

# Removing an element from the stack

def pop(stack):

    if (is\_empty(stack)):

        return "The stack is empty"

    return stack.pop()

stack = create\_stack()

push(stack, str(1))

push(stack, str(2))

push(stack, str(3))

push(stack, str(4))

push(stack, str(5))

print("The elements in the stack are:"+ str(stack))

Answer the following questions:

1. What is the main difference between the stack and queue implementations in terms of element removal?
2. What would happen if we try to dequeue from an empty queue, and how is this handled in the code?
3. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?
4. What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?
5. In real-world applications, what are some practical use cases where queues are preferred over stacks?

# **Results**

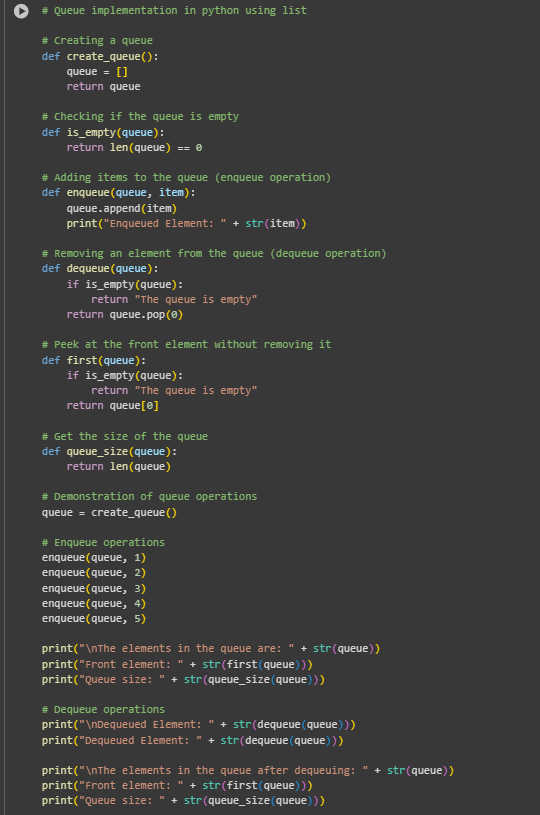


Figure 1: It shows the stack revised into queue’s

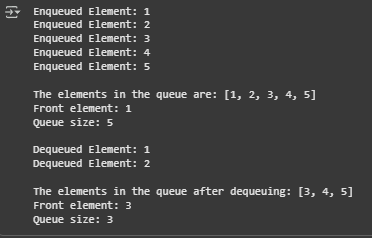


Figure 2: It shows the output of the program using queue’s

**1. What is the main difference between the stack and queue implementations in terms of element removal?**

* Stack: Like a stack of plates - you take from the top (last one you put)
* Queue: Like a line at a store - first person in line gets served first

**2. What would happen if we try to dequeue from an empty queue, and how is this handled in the code?**

If you try to remove from an empty queue, our code shows "The queue is empty" message. This prevents the program from crashing

**3. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?**

If we add new items at the front instead of back, it becomes like a stack. And, new items would be removed first instead of old items. This breaks the "first come, first served" rule.

**4.  What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?**

***Linked List:***  
***Pros:***

* Fast removal from front
* Grows easily without copying

**Cons:**

* Uses more memory
* Slower to access elements

***Array:***  
***Pros:***

* Uses less memory
* Faster to access elements

**Cons:**

* Slow removal from front
* Needs copying when full

**5.  In real-world applications, what are some practical use cases where queues are preferred over stacks?**

* Task scheduling in operating systems
* Print job management
* Breadth-first search algorithms
* Message queues in distributed systems
* Customer service systems (call centers)
* Network packet routing

# Conclusion

In conclusion, this laboratory successfully implemented a queue in Python, demonstrating its FIFO behavior and key operations like enqueue and dequeue. We learned that queues process elements in arrival order, making them ideal for real-world applications like task scheduling and customer service systems where fair, sequential processing is essential.

**References**

[1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.

[2] A. O. Co, "University of Caloocan City Computer Engineering Department Honor Code," UCC-CpE Departmental Policies, 2020.

[3] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Data Structures and Algorithms in Python. Hoboken, NJ: John Wiley & Sons, 2013.

[4] D. E. Knuth, The Art of Computer Programming, Volume 1: Fundamental Algorithms, 3rd ed. Boston, MA: Addison-Wesley, 1997.