UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Regondola, Jezreel P.

Instructor:
Engr. Maria Rizette H. Sayo

October 11, 2025

I. 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

II. 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))
```

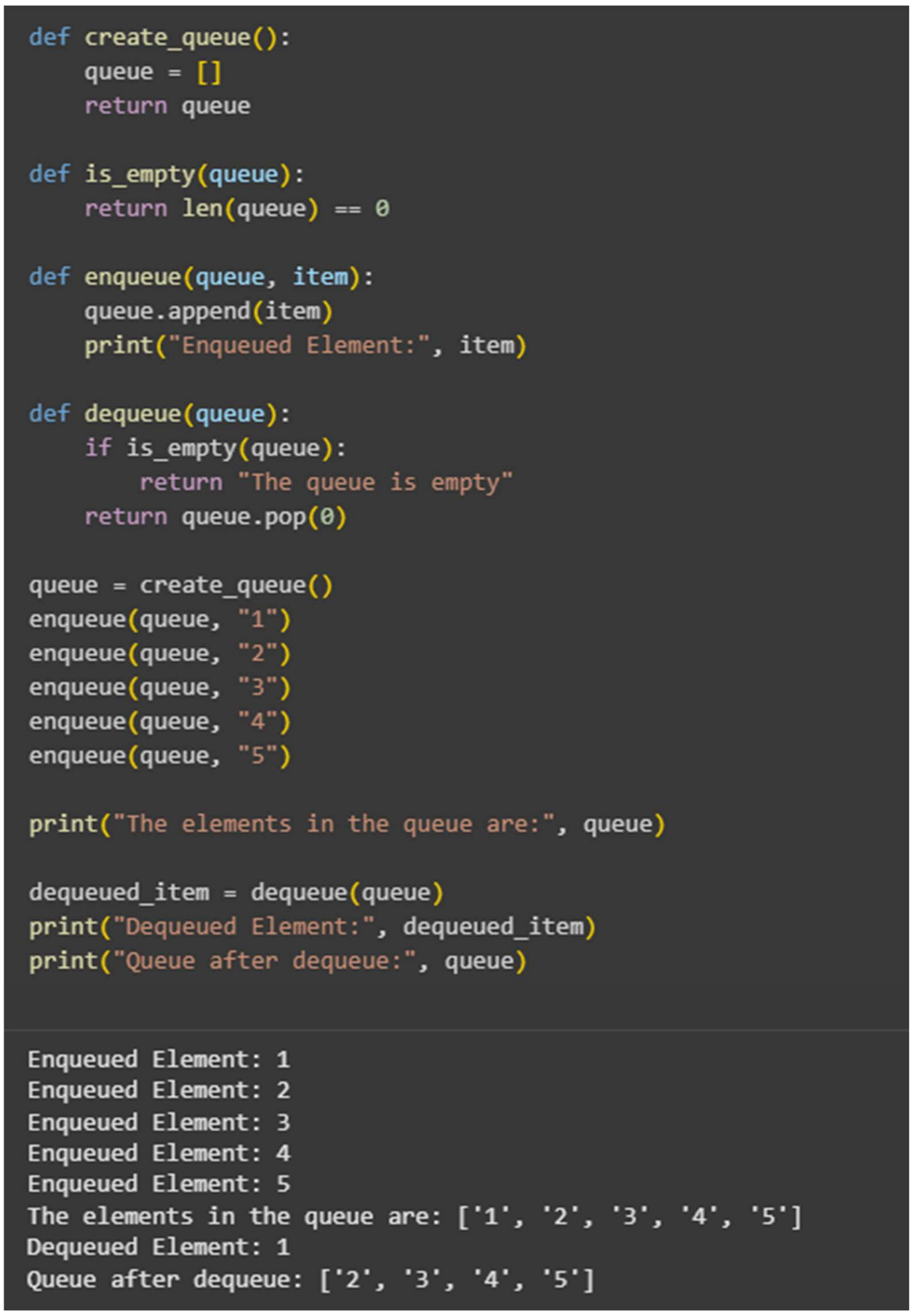
Answer the following questions:

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?

III. Results

1. A stack removes the most recently added element (LIFO), while a queue removes the element that has been in the structure the longest (FIFO).
2. Attempting to dequeue from an empty queue would result into an error or message like “The queue is empty”. In the code, this is handled by checking if the queue is empty before performing the dequeue operation.
3. This would reverse the queue’s behavior and make it like a stack, where the most recently added item gets removed first.
4. Linked lists allow us to allocate memory dynamically and insert/delete faster but it requires more memory. While arrays are easier to implement and have faster access to elements but resizing them can be inefficient and time consuming.

5. Queues are used in CPU process management, customer service systems, data streaming, and message queues in communication systems.



```python
def create_queue():
    queue = []
    return queue

def is_empty(queue):
    return len(queue) == 0

def enqueue(queue, item):
    queue.append(item)
    print("Enqueued Element:", item)

def dequeue(queue):
    if is_empty(queue):
        return "The queue is empty"
    return queue.pop(0)

queue = create_queue()
enqueue(queue, "1")
enqueue(queue, "2")
enqueue(queue, "3")
enqueue(queue, "4")
enqueue(queue, "5")

print("The elements in the queue are:", queue)

dequeued_item = dequeue(queue)
print("Dequeued Element:", dequeued_item)
print("Queue after dequeue:", queue)
```

```
Enqueued Element: 1
Enqueued Element: 2
Enqueued Element: 3
Enqueued Element: 4
Enqueued Element: 5
The elements in the queue are: ['1', '2', '3', '4', '5']
Dequeued Element: 1
Queue after dequeue: ['2', '3', '4', '5']
```

Figure 1 Screenshot of program

IV. Conclusion

In this activity, I learned how to implement and use the queue data structure in Python. I was able to perform enqueue and dequeue operations and understand how the FIFO principle works. This improved my understanding of how queues differ from stacks and how they are applied in real-world systems like in browser history and undo functions.

References

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