UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Mamanao Kurt Marwin C.

Instructor:
Engr. Maria Rizette H. Sayo

Oct, 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))
```

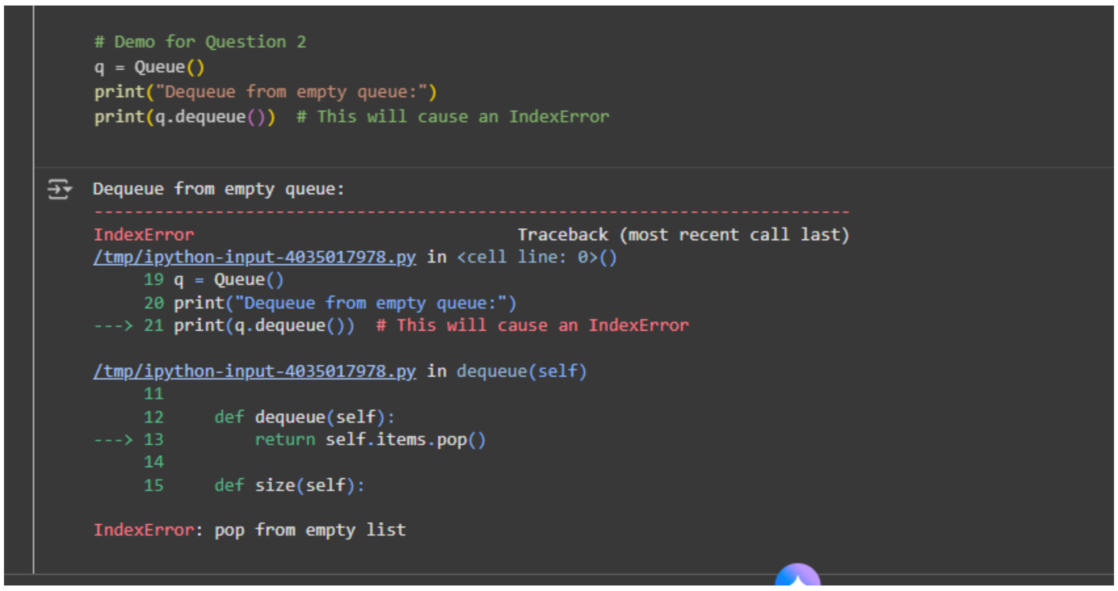
Answer the following questions:

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

What is the main difference in removing elements from a stack vs a queue?
- In a stack, elements are removed from the **top** (LIFO), while in a queue, they are removed from the **front** (FIFO).

What happens if we try to dequeue from an empty queue?



```
# Demo for Question 2
q = Queue()
print("Dequeue from empty queue:")
print(q.dequeue())  # This will cause an IndexError
```

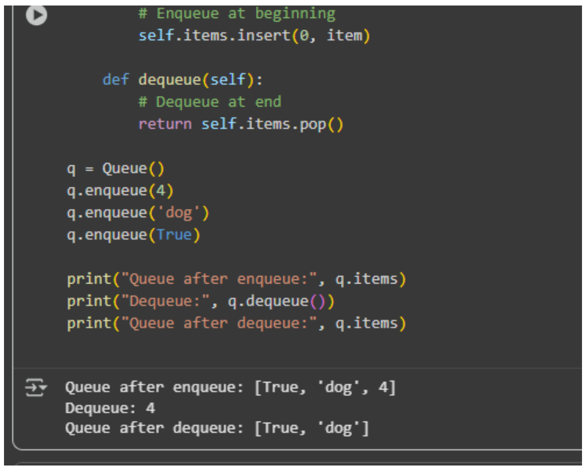
```
Dequeue from empty queue:
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipython-input-4035017978.py in <cell line: 0>()
     19 q = Queue()
     20 print("Dequeue from empty queue:")
---> 21 print(q.dequeue())  # This will cause an IndexError

/tmp/ipython-input-4035017978.py in dequeue(self)
     11
     12     def dequeue(self):
---> 13         return self.items.pop()
     14
     15     def size(self):

IndexError: pop from empty list
```

- When I tried to dequeue from an empty queue, the program gave an error because there's nothing to remove. The code didn't handle it, so it just crashed. If I want to fix it, I need to check if the queue is empty before removing anything.

If enqueue is at the beginning instead of the end, what will happen?



```
        # Enqueue at beginning
        self.items.insert(0, item)

    def dequeue(self):
        # Dequeue at end
        return self.items.pop()

q = Queue()
q.enqueue(4)
q.enqueue('dog')
q.enqueue(True)

print("Queue after enqueue:", q.items)
print("Dequeue:", q.dequeue())
print("Queue after dequeue:", q.items)
```

```
Queue after enqueue: [True, 'dog', 4]
Dequeue: 4
Queue after dequeue: [True, 'dog']
```

- If I enqueue at the beginning and still dequeue at the end, it will act like a **stack**. The last item I put in will be the first one to come out. It's no longer FIFO.

What are the advantages and disadvantages of implementing a queue using a linked list vs an array?

-When I tried to dequeue from an empty queue, the program gave an error because there's nothing to remove. The code didn't handle it, so it just crashed. If I want to fix it, I need to check if the queue is empty before removing anything.

Give real-world examples where queues are preferred.
-Some real life examples are lines in fast food, printer jobs, task scheduling, and waiting lists in apps. The first one who comes in is the first one who gets served, just like a queue.

IV. Conclusion

In this activity, I learned how to change a stack into a queue. At first, it was a bit confusing because stack and queue work differently, but while doing the activity, I started to understand FIFO. I also learned what happens if I try to dequeue when the queue is empty and why the position of enqueue and dequeue matters. This lesson helped me remember our past topics and understand queues better.

References

[1] GeeksforGeeks, “Queue Data Structure,” *GeeksforGeeks*, 2024. [Online]. Available: https://www.geeksforgeeks.org/queue-data-structure/