



## Data Structure and Algorithm

### Laboratory Activity No. 9

---

# Queues

---

*Submitted by:*  
Acebedo, Sebastian C.

*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))

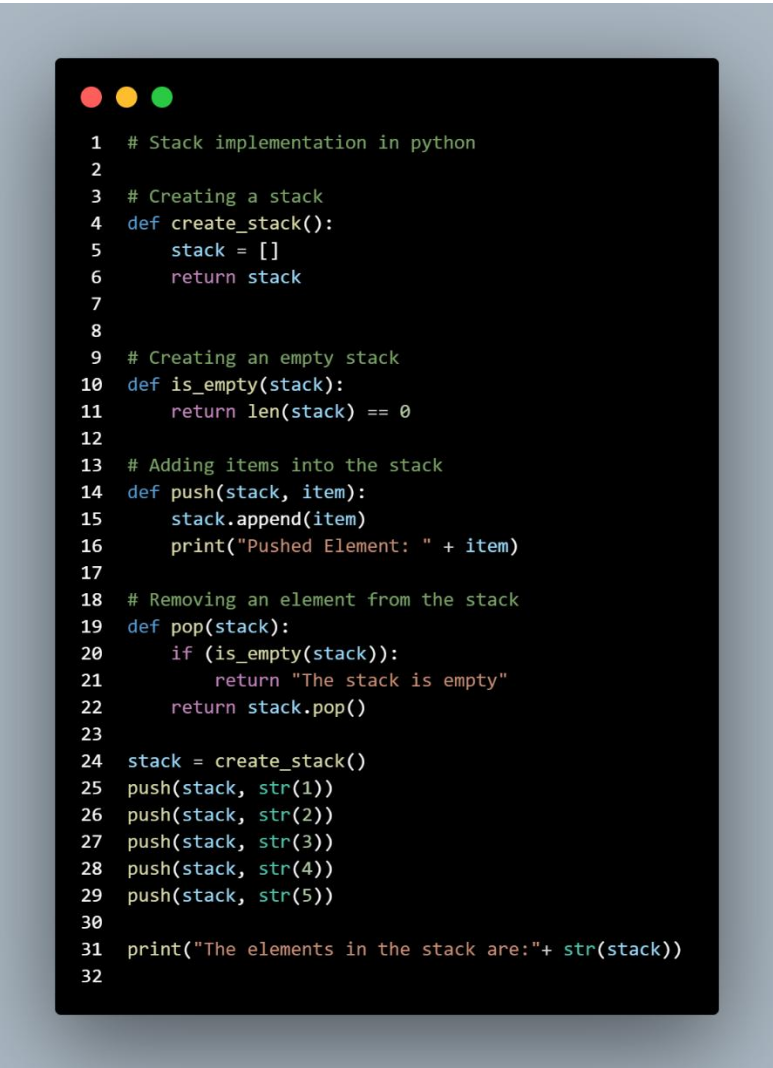
```

Answer the following questions:

- 1 What is the main difference between the stack and queue implementations in terms of element removal?
  - In a stack, elements are removed from the top, which means the last element added is the first one to be removed. This follows the LIFO (Last In, First Out) rule. On the other hand, in a queue, elements are removed from the front, so the first element added is the first one removed. This follows the FIFO (First In, First Out) rule.
- 2 What would happen if we try to dequeue from an empty queue, and how is this handled in the code?
  - If we try to remove an element from an empty queue, the program will check first if it's empty. If it is, it will show a message like "The queue is empty" instead of crashing. This prevents errors in the program.
- 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 modify the enqueue operation to add elements at the beginning instead of the end, the queue will start working like a stack. This is because the last element added will be the first one removed, which follows the LIFO (Last In, First Out) rule instead of FIFO (First In, First Out).
- 4 What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?

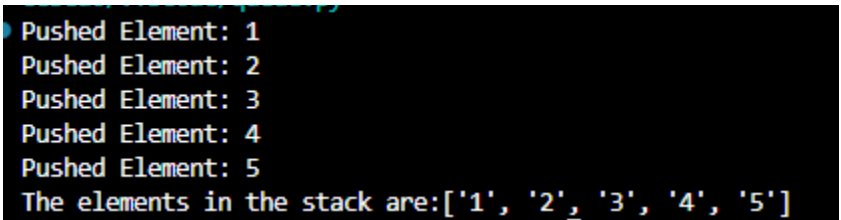
- Using an array for a queue is easy and fast to use, but it has a fixed size and can get full quickly. A linked list is better because it can grow or shrink anytime, but it uses more memory and can be a little slower since it has pointers.
- 5 In real-world applications, what are some practical use cases where queues are preferred over stacks?
- Queues are used in real life when things need to be done in order, like when ordering at McDonald's. The first person who orders is the first one to get their food, while the others wait for their turn. That's why queues follow the FIFO (First In, First Out) rule and are used more than stacks when order is important.

### III. Results

A screenshot of a code editor with a dark background and light-colored text. The code is a Python script for implementing a stack. It includes comments for each function: 'create\_stack', 'is\_empty', 'push', and 'pop'. The 'push' function appends items to a list and prints them. The 'pop' function checks if the stack is empty before removing an item. At the bottom, the script creates a stack, pushes five elements (1-5), and prints the final stack contents.

```
1 # Stack implementation in python
2
3 # Creating a stack
4 def create_stack():
5     stack = []
6     return stack
7
8
9 # Creating an empty stack
10 def is_empty(stack):
11     return len(stack) == 0
12
13 # Adding items into the stack
14 def push(stack, item):
15     stack.append(item)
16     print("Pushed Element: " + item)
17
18 # Removing an element from the stack
19 def pop(stack):
20     if (is_empty(stack)):
21         return "The stack is empty"
22     return stack.pop()
23
24 stack = create_stack()
25 push(stack, str(1))
26 push(stack, str(2))
27 push(stack, str(3))
28 push(stack, str(4))
29 push(stack, str(5))
30
31 print("The elements in the stack are:" + str(stack))
32
```

Figure 1 Screenshot of program

A screenshot of a terminal window showing the output of the Python script. It displays five lines of 'Pushed Element' messages followed by a final line showing the stack as a list of strings.

```
Pushed Element: 1
Pushed Element: 2
Pushed Element: 3
Pushed Element: 4
Pushed Element: 5
The elements in the stack are:['1', '2', '3', '4', '5']
```

Figure 2 Screenshot of program

## IV. Conclusion

In conclusion, a queue is important because it helps organize tasks and process things in the right order, like in lines or scheduling systems. It follows the FIFO rule, where the first element added is the first one removed. Queues are widely used in programming to manage data efficiently, such as in task scheduling, handling requests, or processing messages in the correct order. They ensure fairness when multiple tasks or items need to be handled one by one. In real life, queues can be seen in places like banks, restaurants, or print lines, where the first person to arrive is served first. Overall, queues are very useful because they keep things organized, ensure tasks are done in order, and make both programming and daily activities run smoothly.

## References

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