UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 8

# Stacks

*Submitted by:*
Delinia, Filjohn B.

*Instructor:*
Engr. Maria Rizette H. Sayo

October 4, 2025

# I. Objectives

Introduction

A stack is a collection of objects that are inserted and removed according to the last-in, first-out (LIFO) principle.

A user may insert objects into a stack at any time, but may only access or remove the most recently inserted object that remains (at the so-called “top” of the stack)

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

- Writing Python program using Stack
- Writing a Python program that will implement Stack operations

# II. Methods

Instruction: Type the python codes below in your Colab. After running your codes, answer the questions below.

```
# 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 Upon typing the codes, what is the name of the abstract data type? How is it implemented?
2 What is the output of the codes?
3 If you want to type additional codes, what will be the statement to pop 3 elements from the top of the stack?
4 If you will revise the codes, what will be the statement to determine the length of the stack? (Note: You may add additional methods to count the no. of elements in the stack)

# III. Results

**Anwers:**

1) Stack, it is implemented using a Python list ([]). The append() method is used to push elements (add to the top), and pop() removes the top element, following the LIFO principle.

2)
```
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 1.**

3) To pop the 3 elements from the top of the stack , I use for loop.

**Additional code:**

for _ in range(3):

print(“Popped Element:”, pop(stack))

**Explanation:** This loop runs three times and calls the pop() function each time, removing the topmost element from the stack and printing it. This will remove the last three elements the were pushed onto the stack.

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

4) To determine then length of the stack, I use this code.

**Additional code:**

print(“Current stack size; “ + str(size(stack)))

**Explanation:** This prints the number of elements currently in the stack.

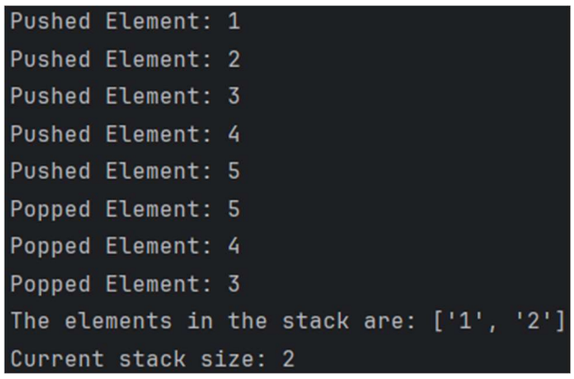


**Figure 3.**

# IV. Conclusion

This exercise has given me a clear knowledge of how a stack functions as a fundamental data structure in Python. I used the Last In, First Out (LIFO) principle to manage elements by implementing fundamental functions like push, pop, and is_empty. By adding a size function to the program, I was able to determine the number of elements now stored in the stack, which provided more functionality and insight. Transitioning to a class-based framework improved the code's organization and scalability, making it more modular and maintainable.

**Source Code:**

**Question 3.**

```
# 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))

for _ in range(3):
    print("Popped Element:", pop(stack))

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

**Question 4.**

```
# Creating a stack
def create_stack():
    stack = []
    return stack

# Checking if the stack is empty
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()

# Getting the size of the stack
def size(stack):
    return len(stack)

# Main program
stack = create_stack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
push(stack, str(5))

for _ in range(3):
    print("Popped Element: " + str(pop(stack)))

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

print("Current stack size: " + str(size(stack)))
```

# References

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

[2] Python Software Foundation.*5. Data Structures*. Python.org. https://docs.python.org/3/tutorial/datastructures.html

[3] GeeksforGeeks. *Stack Data Structure*. Retrieved October 4, 2025, https://www.geeksforgeeks.org/stack-data-structure/

[4] Programiz. (n.d.). *Stack in Python*. Retrieved October 4, 2025, https://www.programiz.com/dsa/stack