



UNIVERSITY OF CALOOCAN CITY  
COMPUTER ENGINEERING DEPARTMENT



Data Structure and Algorithm

Laboratory Activity No. 8

---

# Stacks

---

*Submitted by:*

Luminario, Venice Lou Gabrielle M.

*Instructor:*

Engr. Maria Rizette H. Sayo

Oct 04, 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?

The code implements an abstract data type known as a Stack.

- 2 What is the output of the codes?

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

- 3 If you want to type additional codes, what will be the statement to pop 3 elements from the top of the stack?

To pop 3 elements, you would add the following lines:

```
print("Popped:", pop(stack))
print("Popped:", pop(stack))
print("Popped:", pop(stack))
```

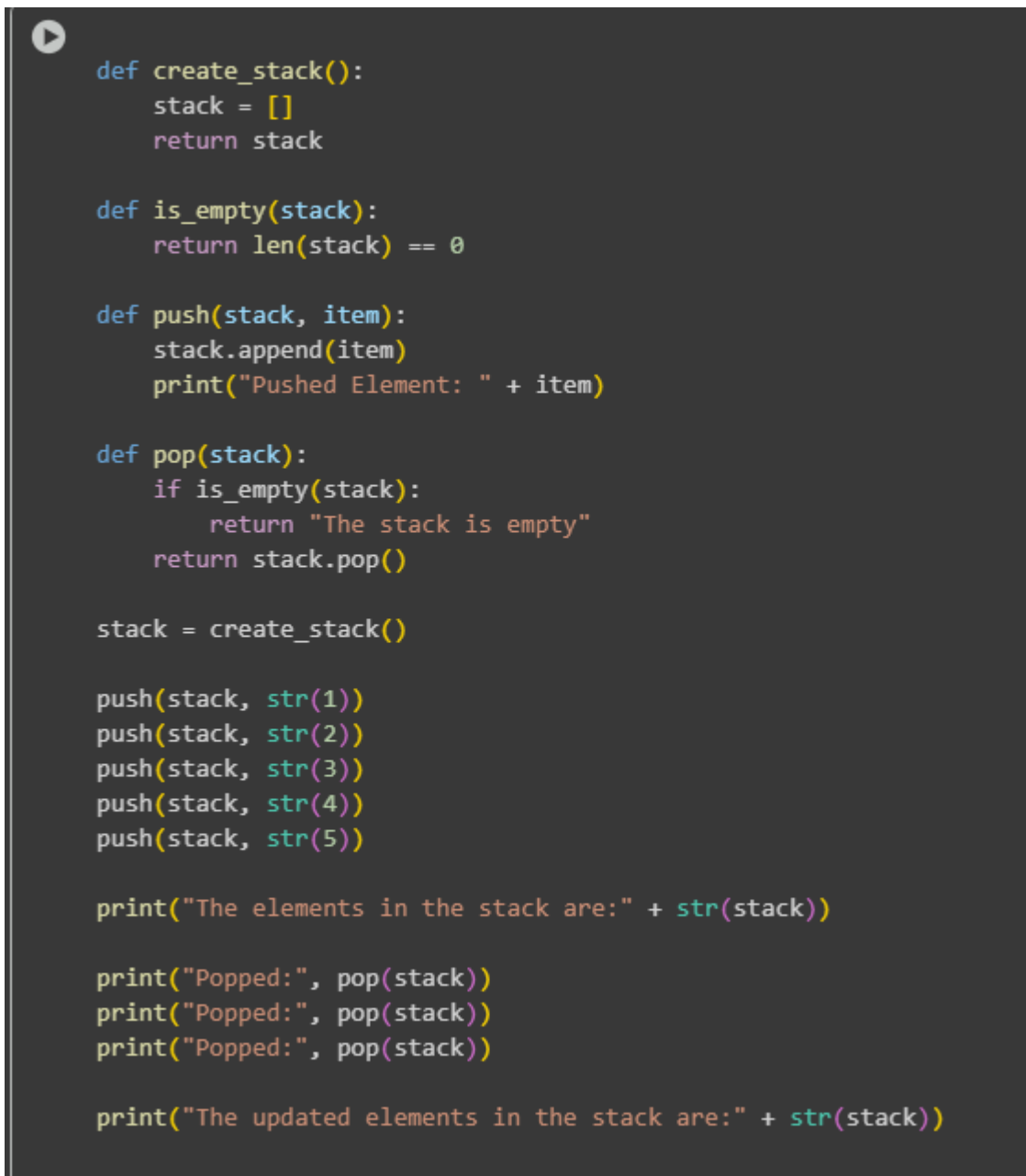
- 4 If you will revise the codes, what will be the statement to determine the length of the stack? (Note: You may add additional methods to count the no. of elements in the stack)

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

### III. Results

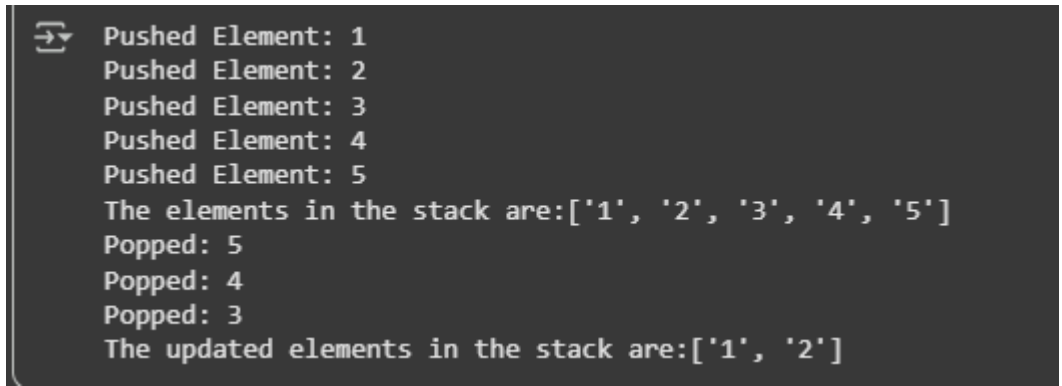
Present the visualized procedures done. Also present the results with corresponding data visualizations such as graphs, charts, tables, or image . Please provide insights, commentaries, or explanations regarding the data. If an explanation requires the support of literature such as academic journals, books, magazines, reports, or web articles please cite and reference them using the IEEE format.

Please take note of the styles on the style ribbon as these would serve as the style format of this laboratory report. The body style is Times New Roman size 12, line spacing: 1.5. Body text should be in Justified alignment, while captions should be center-aligned. Images should be readable and include captions. Please refer to the sample below:

A screenshot of a code editor with a dark background. The code is written in Python and implements a stack using a list. It includes functions for creating a stack, checking if it's empty, pushing elements, and popping elements. The main execution part creates a stack, pushes five elements (1-5), prints the stack, pops three elements, and prints the updated stack. A play button icon is visible in the top left corner of the code editor.

```
def create_stack():  
    stack = []  
    return stack  
  
def is_empty(stack):  
    return len(stack) == 0  
  
def push(stack, item):  
    stack.append(item)  
    print("Pushed Element: " + item)  
  
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))  
  
print("Popped:", pop(stack))  
print("Popped:", pop(stack))  
print("Popped:", pop(stack))  
  
print("The updated elements in the stack are:" + str(stack))
```

Figure 1 Screenshot of program



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

Figure 2: Output

Conclusion

This is a fantastic learning tool. The way the code is broke down made the code so much easier to read. The 'Last-In, First-Out' concept is perfectly clear here. It really shows that making a stack in Python doesn't have to be complicated.

## References

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