UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 8

Stacks

Submitted by:
Disomnong, Jalilah M.

Instructor:
Engr. Maria Rizette H. Sayo

10/04/2025

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

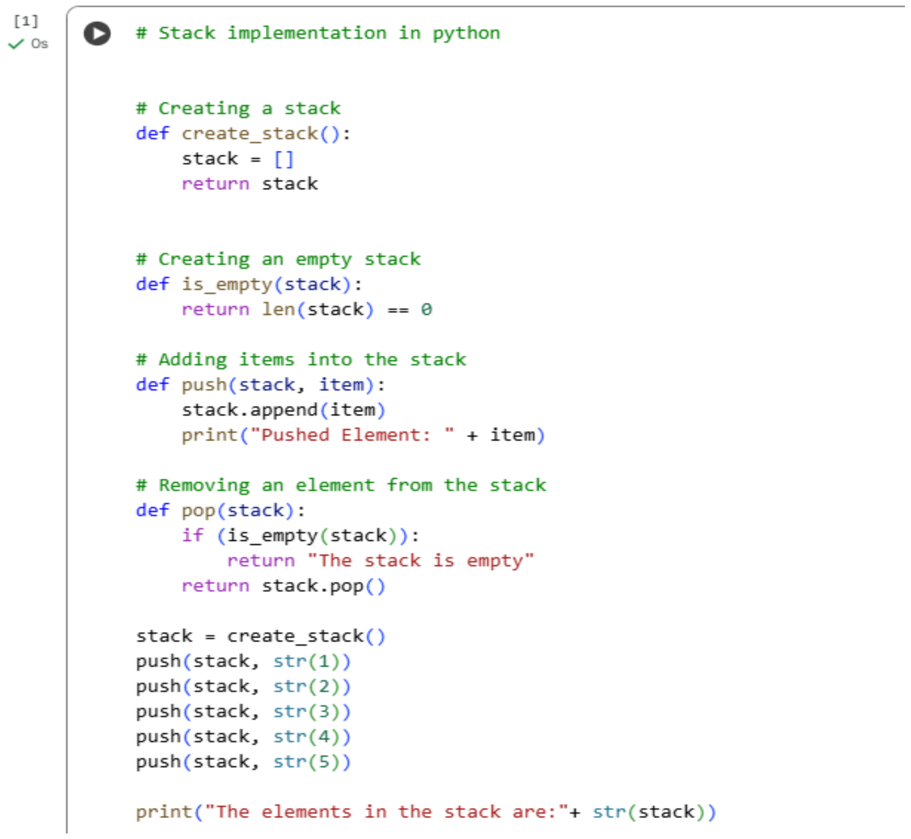
Instruction: Type the python codes below in your Colab. After running your codes, answer the questions below.

```
# 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?
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)

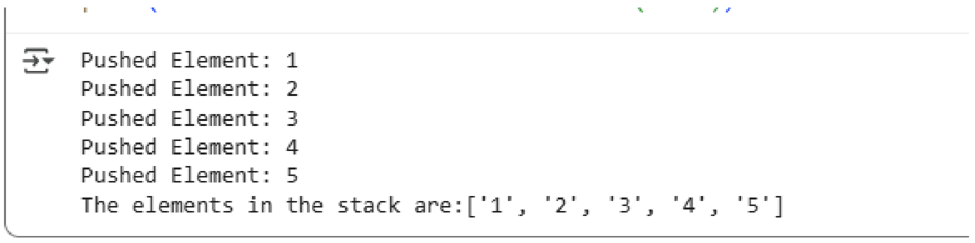
III. Results



```
[1]
✓ 0s
# 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))
```

Figure 1 Screenshort of the source in Google Colab https://colab.research.google.com/drive/1nqwSpz6gizLYqQlGy9M-1ZGGofmxCKhC?authuser=0#scrollTo=Fu1TpDWXS8g9

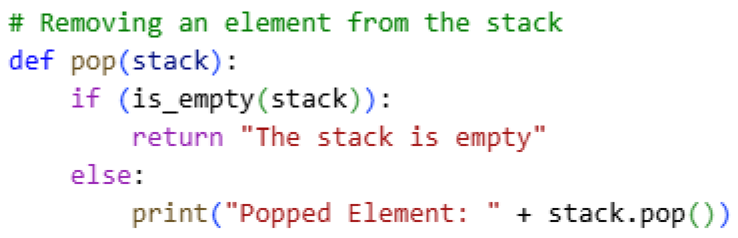
The data type used in this was an abstract data type called a stack, implemented using a Python list. The append() method adds (pushes) elements to the stack, while pop()removes the most recent element, following the Last In, First Out (LIFO) principle.



```
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 Output of the source code

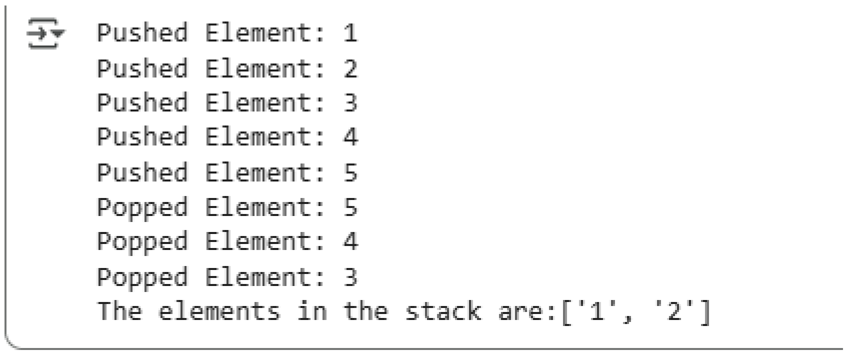
The output of the code shows the process of pushing five elements onto a stack and then printing the current contents of the stack. Each time an element is pushed, a message is printed to indicate which element was added. The elements pushed are the string versions of numbers 1 through 5. After pushing, the code prints the full stack, which will display as ['1', '2', '3', '4', '5']



```python
# Removing an element from the stack
def pop(stack):
    if (is_empty(stack)):
        return "The stack is empty"
    else:
        print("Popped Element: " + stack.pop())
```

Figure 3 Screenshot of Adding sourcecode to pop an element

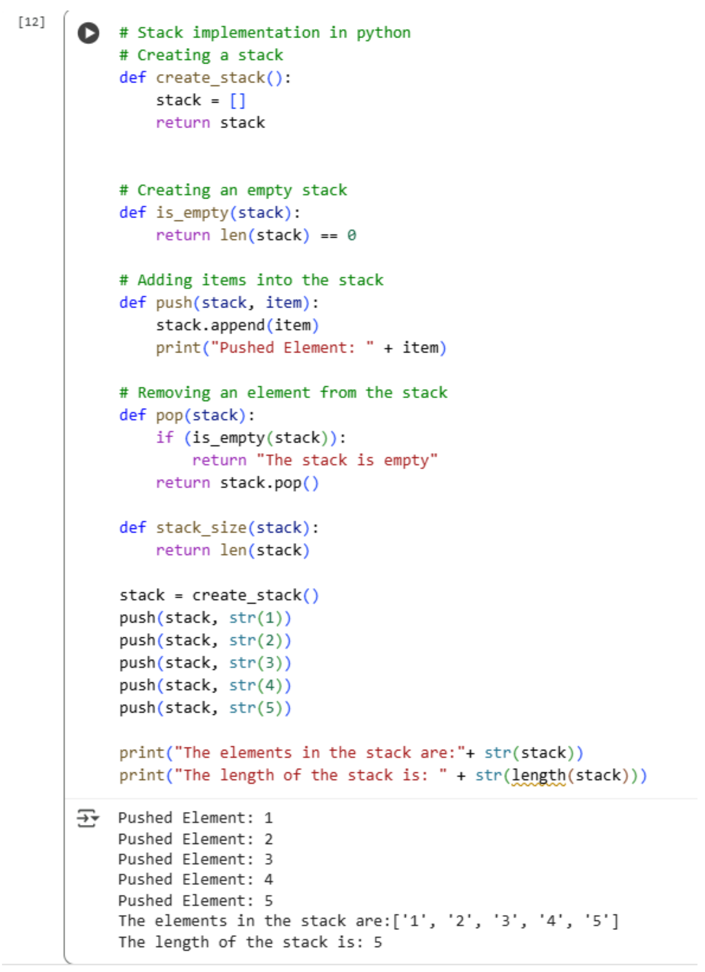
Using the else statement, three elements can be popped from the top of the stack by calling the pop() function three times. The else ensures that elements are only removed when the stack is not empty, avoiding runtime errors. This approach follows the Last In, First Out (LIFO) principle, where the most recently added elements are the first to be removed. After popping three times, the stack contains only the remaining elements.



```
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']
```

Figure 4 Output of the popping 3 elements

To revise the code to determine the length of the stack, we can add a method like stack_size() that returns len(stack). This method counts the number of elements in the stack and keeps the code organized by separating this logic from other stack operations.



```
[12]
# 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()

def stack_size(stack):
    return len(stack)

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("The length of the stack is: " + str(length(stack)))
```

```
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']
The length of the stack is: 5
```

Figure 5 Screenshot of adding a method of len() with the output of sourcecode

IV. Conclusion

In this lab, it was demonstrated how to create and use a stack in Python based on the Last In, First Out (LIFO) principle. The process of adding (pushing) and removing (popping) elements was practiced, showing how the stack operates step-by-step. This activity helped provide a better understanding of stack behavior and its practical implementation in programming.

References

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

[2] Google. (2024). Google Colaboratory. Retrieved April 18, 2024, from https://colab.research.google.com/