Namal University, Mianwali
Department of Electrical Engineering
Course Title: EE-253 Data Structures and Algorithms

LAB MANUAL

Lab 6 Stacks in Python

In previous lab, we have studied the concept of linked lists in python. We have studied types of linked lists (singly linked list, circular linked list and doubly linked list), how to make a linked list and perform different operations (traversing, updating, removing, etc.) on linked list.

1. Lab Objectives

The objective of this lab is to introduce students to the concept of stacks in python. In this lab students will enable the concepts of implementing a stack, accessing the elements and operations which can be performed on stack. Students will be provided with examples, followed by performing lab tasks.

2. Lab Outcomes

- CLO:1 Recognize the usage of fundamental Data structure using Python Programming Language.
- CLO:3 Demonstrate solution to real life problems using appropriate data structures.

3. Equipment

- Software
    - IDLE (Python 3.11)

4. Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as ‘Example’) are for practice purposes only while others (marked as ‘Task’) have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
    a. Code (copy and pasted, NOT a screenshot)
    b. Output figure (as instructed in 3)
    c. Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly. Any incident where plagiarism is caught, both (or all) students involved will be given zero marks, regardless of who copied whom. Multiple such incidents will result in disciplinary action being taken.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

5. Background

Stack:
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). The name "stack" is derived from the metaphor of a stack of plates in a spring-loaded, cafeteria plate dispenser. In this case, the fundamental operations involve the **"pushing"** and **"popping"** of plates on the stack. When we need a new plate from the dispenser, we **"pop"** the top plate off the stack, and when we add a plate, we **"push"** it down on the stack to become the new top plate. Perhaps an even more amusing example is a PEZ ® candy dispenser, which stores mint candies in a spring-loaded container that "pops" out the topmost candy in the stack when the top of the dispenser is lifted as shown in below figure.

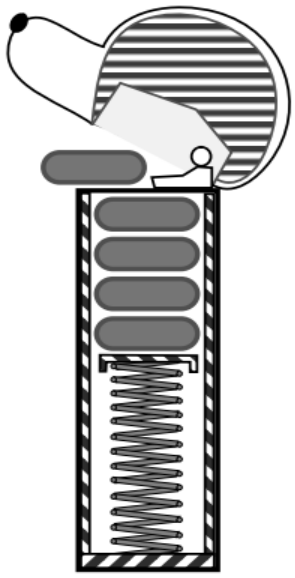


Figure 1: A schematic drawing of a PEZ ® dispenser; a physical implementation of the stack ADT.

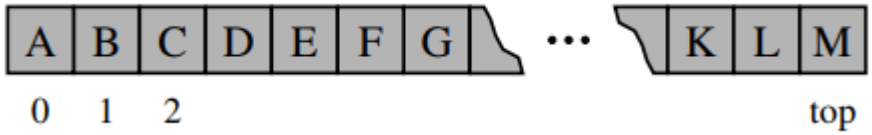
Stacks are a fundamental data structure. They are used in many applications. For example, Internet Web browsers store the addresses of recently visited sites in a stack. Each time a user visits a new site, that site's address is "pushed" onto the stack of addresses. The browser then allows the user to "pop" back to previously visited sites using the "back" button. Another example, text editors usually provide an "undo" mechanism that cancels recent editing operations and reverts to former states of a document. This undo operation can be accomplished by keeping text changes in a stack.

Stack Abstract Data Type (Methods and Operations):
Stacks are the simplest of all data structures, yet they are also among the most important. They are used in a host of different applications, and as a tool for many more sophisticated data structures and algorithms. Formally, a stack is an abstract data

type (ADT) such that an instance S supports the following two methods:
S.push(e): Add element e to the top of stack S.
S.pop( ): Remove and return the top element from the stack S;
an error occurs if the stack is empty.
Additionally, let us define the following accessor methods for convenience:
S.top( ): Return a reference to the top element of stack S, without removing it; an error occurs if the stack is empty.
S.is empty( ): Return True if stack S does not contain any elements.
len(S): Return the number of elements in stack S; in Python, we implement this with the special method len .
By convention, we assume that a newly created stack is empty, and that there is no a priori bound on the capacity of the stack. Elements added to the stack can have arbitrary type.
Here are some basic methods applied on stack as shown in below table.

Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop( )	3	[5]
S.is_empty( )	False	[5]
S.pop( )	5	[ ]
S.is_empty( )	True	[ ]
S.pop( )	"error"	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top( )	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop( )	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop( )	8	[7, 9, 6]

Implementation of Stack:
We can implement a stack quite easily by storing its elements in a Python list. The list class already supports adding an element to the end with the append method, and removing the last element with the pop method, so it is natural to align the top of the stack at the end of the list, as shown in below figure.



Example 1: Below, we present an example of the use of our ArrayStack class to implement stack. In this example we have implemented two important methods (push and pop) of stacks which are mostly used. We have also implemented some other methods as well.

Source code:

```
class ArrayStack:
  # LIFO Stack implementation using a Python list as underlying storage.
  def __init__(self):
    # Create an empty stack.
    self._data = []  # nonpublic list instance

  def __len__(self):
    # Return the number of elements in the stack.
    return len(self._data)

  def is_empty(self):
    # Return True if the stack is empty.
    return len(self._data) == 0

  def push(self, e):
    # Add element e to the top of the stack.
    self._data.append(e)  # new item stored at end of list

  def top(self):
    '''Return (but do not remove) the element at the top of the stack.
    Raise Empty exception if the stack is empty.
    '''
    if self.is_empty():
      raise Empty('Stack is empty')
    return self._data[-1]  # the last item in the list

  def pop(self):
    '''Remove and return the element from the top of the stack (i.e., LIFO).
    Raise Empty exception if the stack is empty.
    '''
    if self.is_empty():
      raise Empty('Stack is empty')
    return self._data.pop()  # remove last item from list
```

```
# Create a new stack instance
stack = ArrayStack()

# Push some elements onto the stack
stack.push(100)
stack.push(200)
stack.push(300)

# Print the length of the stack (should be 3)
print(len(stack))

# Print the top element of the stack (should be 3)
print(stack.top())

# Pop the top element off the stack (should return 3)
print(stack.pop())

# Print the length of the stack (should be 2)
print(len(stack))

# Pop the remaining elements off the stack (should return 2)
print(stack.pop())

# Print the length of the stack (should be 1)
print(len(stack))

# Pop the remaining elements off the stack (should return 2)
print(stack.pop())

# Print whether the stack is empty (should be True)
print(stack.is_empty())
```

Output:

```
3
300
300
2
200
1
100
True
```

6. Lab Tasks:

Task 1: A shop has a stack of chocolate boxes each containing a positive number of chocolates. Initially, the stack is empty. During the next N minutes, either of these two things may happen:
The box of chocolates on top of the stack gets sold. Make a method (chocolates_sold)
You receive a box of chocolates from the warehouse and put it on top of the stack. Make a method (chocolates_available)
Determine the number of chocolates in the sold box each time he sells a box. Also determine the number of chocolates which are available in the stack.

Task 2: Alice is rearranging her library. She takes the innermost shelf and reverses the order of books. She breaks the walls of the shelf. In the end, there will be only books and no shelf walls. Print the order of books.
Opening and closing walls of shelves are shown by '/' and '\' respectively whereas books are represented by lower case alphabets. Implement it using stack in python.

Home Task : A and B are playing a game. In this game, both of them are initially provided with a list of numbers. (Both have the same list but their own copy.)
Now, they both have a different strategy to play the game.
A picks the element from start of his list. B picks from the end of his list.
You need to generate the result in form of an output list.
Method to be followed at each step to build the output list is:
If the number picked by A is bigger than B, then this step's output is 1. B removes the number that was picked from their list.
If the number picked by A is smaller than B, then this step's output is 2. A removes the number that was picked from their list.
If both have the same number, then this step's output is 0. Both A and B remove the number that was picked from their list.
This game ends when at least one of them has no more elements to be picked i.e. when the list gets empty.
Make a method to play this game. Implement it using stack in python.

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-2**	**Marginal 3-5**	**Satisfactory 6-8**	**Exemplary 9-10**	**Allocated Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested by others. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend for the instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around. More than two incidents of talking non-lab related stuff in laband/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff in lab Voice level exceeding theappropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceedingthe appropriate level, use of cell phones and involvementin any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Tookcare of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	