NAMAL UNIVERSITY MIANWALI

DEPARTMENT OF ELECTRICAL ENGINEERING

DATA STRUCTURE AND ALGORITHM

LAB # 06

REPORT

Title : Stacks in Python

Name	Fahim-Ur-Rehman Shah
Roll No	NIM-BSEE-2021-24
Intructor	Ms. Naureen Shaukat
Lab Engineer	Mr .Ali Hasnain
Date	08-May-2023
Marks	

Instructions:

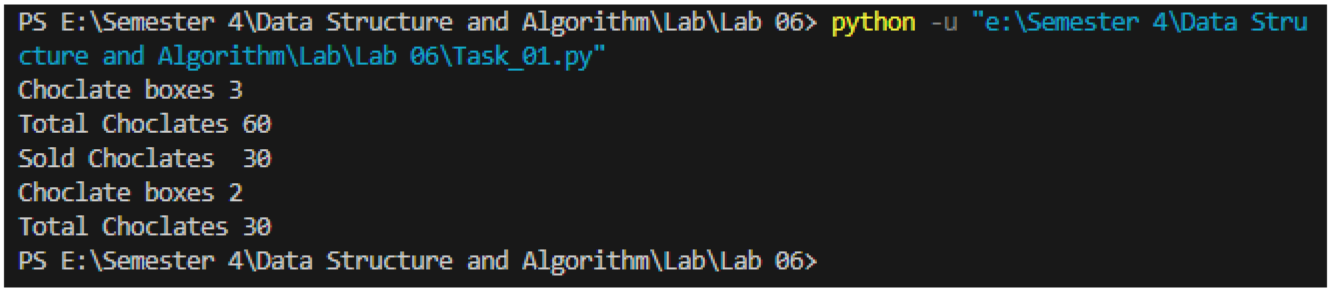
1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as 'Example') are for practice purposes only while others (marked as 'Task') have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg 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).

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.

Python Code :

```python
class ChocolateStack:
    def __init__(self):
        self.stack = []
    def chocolates_sold(self):
        if len(self.stack) == 0:
            return None
        else:
            return self.stack.pop()
    def chocolates_available(self, n_chocolates):
        self.stack.append(n_chocolates)
    def get_total_chocolates(self):
        return sum(self.stack)
    def get_number_of_boxes(self):
        return len(self.stack)
chocolate_stack = ChocolateStack()
chocolate_stack.chocolates_available(10)
chocolate_stack.chocolates_available(20)
chocolate_stack.chocolates_available(30)
print("Choclate boxes",chocolate_stack.get_number_of_boxes()) # 3
print("Total Choclates",chocolate_stack.get_total_chocolates()) # 60
sold_chocolates = chocolate_stack.chocolates_sold()
print("Sold Choclates ",sold_chocolates) # 30
print("Choclate boxes",chocolate_stack.get_number_of_boxes()) # 2
print("Total Choclates",chocolate_stack.get_total_chocolates()) # 40
```

Output Screenshot:



```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 06> python -u "e:\Semester 4\Data Stru
cture and Algorithm\Lab\Lab 06\Task_01.py"
Choclate boxes 3
Total Choclates 60
Sold Choclates  30
Choclate boxes 2
Total Choclates 30
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 06>
```

Explanation :

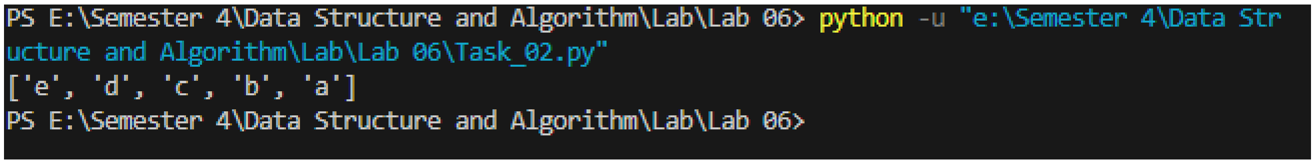
This code defines a class called "ChocolateStack" with methods for adding chocolates to the stack, selling chocolates, and getting information about the stack, such as the total number of chocolates and the number of boxes. The code creates an instance of this class and adds three boxes of chocolates, totaling 60 chocolates, to the stack. The code then sells one box of chocolates, returning the number of chocolates in the sold box, and prints the updated information about the stack, which now has two boxes and 40 chocolates.

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.

Python Code :

```python
class BookStack:
    def __init__(self):
        self.stack = []
    def add_book(self, book):
        self.stack.append(book)
    def reverse_order(self):
        reversed_books = []
        while self.stack:
            reversed_books.append(self.stack.pop())
        return reversed_books
# Example usage:
shelf = "/abcde\\"
book_stack = BookStack()
# Add books to the stack
for book in shelf[1:-1]:
    book_stack.add_book(book)
# Print the reversed order of books
print(book_stack.reverse_order())
```

Output Screenshot:



```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 06> python -u "e:\Semester 4\Data Str
ucture and Algorithm\Lab\Lab 06\Task_02.py"
['e', 'd', 'c', 'b', 'a']
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 06>
```

Explanation :

In this code, we define a class called BookStack which has methods for adding books to the stack and reversing the order of the books. We create an instance of the class and add each book on the innermost shelf to the stack. Finally, we call the reverse_order method to print the reversed order of the books.

Task 3: 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.

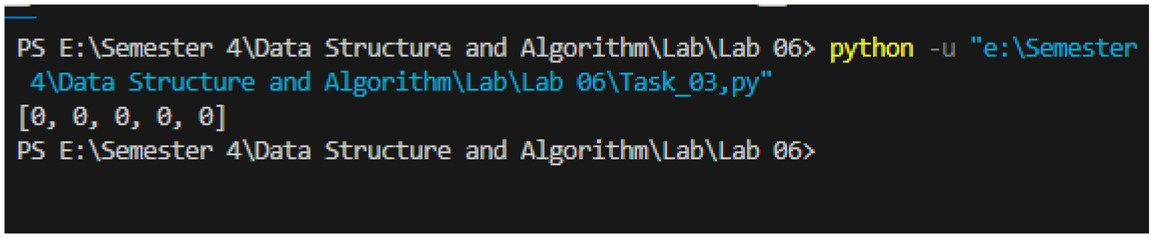
Python Code :

```python
class Game:
    def __init__(self, list_a, list_b):
        self.stack_a = list(reversed(list_a))
        self.stack_b = list(reversed(list_b))
        self.output_list = []

    def play(self):
        while self.stack_a and self.stack_b:
            a = self.stack_a[-1]
            b = self.stack_b[-1]
            if a > b:
                self.output_list.append(1)
                self.stack_b.pop()
            elif a < b:
                self.output_list.append(2)
                self.stack_a.pop()
            else:
                self.output_list.append(0)
                self.stack_a.pop()
                self.stack_b.pop()
        return self.output_list

# Example usage:
list_a = [1, 2, 3, 4, 5]
list_b = [1, 2, 3, 4, 5]
game = Game(list_a, list_b)
output_list = game.play()
print(output_list)
```

Output Screenshot:



```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 06> python -u "e:\Semester
 4\Data Structure and Algorithm\Lab\Lab 06\Task_03,py"
[0, 0, 0, 0, 0]
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 06>
```

Explanation :

In this code, we define a class called Game which has stacks to represent the list that each player has and an output list to record the results of each step in the game. We create an instance of the class with two lists and then call the play method to run the game. The method compares the top elements of each stack, adds a 1, 2, or 0 to the output list depending on the comparison, removes the top element from the corresponding stack, and continues until one of the stacks is empty. Finally, the method returns the output list. The code prints the output list as an example.

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-5**	**Marginal 5-10**	**Satisfactory 11-15**	**Exemplary 16-20**	**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 byothers. 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 forthe 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 not given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is not given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is not given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around.Chased others, ran, or played 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 inlab. 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.	