Activity #1 (Midterm) Laboratory Activity

Create an implementation of the **Stack Data Structure:**

As a guide, you can use the following description of the **Stack Data Structure**.

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.

```python
class Stack:
    def __init__(stack):
        stack.items = []
    16 usages
    def push(stack, value):
        stack.items.append(value)

    13 usages
    def pop(stack):
        return stack.items.pop() if not stack.is_empty() else None

    5 usages
    def is_empty(stack):
        return len(stack.items) == 0

    2 usages
    def top(stack):
        return stack.items[-1] if not stack.is_empty() else None

    def __len__(stack):
        return len(stack.items)


S = Stack()
X = Stack()
print("Operation - Stack - Return Value")

S.push(5)
print("S.push(5):",S.items)

S.push(3)
print("S.push(3):", S.items)

len(S)
print("len(S):", S.items,len(S))

S.pop()
print("S.pop():",S.items)

S.is_empty()
```

```
S.is_empty()
print("S.is_empty():", S.items,S.is_empty())

S.pop()
print("S.pop():", S.items)

print("S.is_empty():", S.items,S.is_empty())

S.pop()
print("S.pop():", S.items)

S.push(7)
print("S.push(7):", S.items)

S.push(9)
print("S.push(9):", S.items)

S.top()
print("S.top():", S.items, S.top())

S.push(4)
print("S.push(4)", S.items)

len(S)
print("len(S):", S.items, len(S))

S.pop()
print("S.pop():", S.items)

S.push(6)
print("S.push(6)", S.items)

S.push(8)
print("S.push(8)",S.items)

S.pop()
print("S.pop():", S.items)

print()
```

OUTPUT:

```
Operation - Stack - Return Value
S.push(5): [5]
S.push(3): [5, 3]
len(S): [5, 3] 2
S.pop(): [5]
S.is_empty(): [5] False
S.pop(): []
S.is_empty(): [] True
S.pop(): []
S.push(7): [7]
S.push(9): [7, 9]
S.top(): [7, 9] 9
S.push(4) [7, 9, 4]
len(S): [7, 9, 4] 3
S.pop(): [7, 9]
S.push(6) [7, 9, 6]
S.push(8) [7, 9, 6, 8]
S.pop(): [7, 9, 6]
```

Next, simulate the **Stack Data Structure** using the table below:

Operation
S.push(5)
S.push(3)
len(S)
S.pop( )
S.is_empty( )
S.pop( )
S.is_empty( )
S.pop( )
S.push(7)
S.push(9)
S.top( )
S.push(4)
len(S)
S.pop( )
S.push(6)
S.push(8)
S.pop( )

Lastly, simulate the following operations to answer the question listed.

What values are returned during the following series of stack operations, if executed upon an initially empty stack? **push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop()**.

```
print()

X.push(5)
print("push(5)", X.items)

X.push(3)
print("push(3)", X.items)

X.pop()
print("pop():", X.items)

X.push(2)
print("push(2)", X.items)

X.push(8)
print("push(8)", X.items)

X.pop()
print("pop():", X.items)

X.pop()
print("pop():", X.items)

X.push(9)
print("push(9)", X.items)

X.push(1)
print("push(1)", X.items)

X.pop()
print("pop():", X.items)

X.push(7)
print("push(7)", X.items)
```

```
X.push(7)
print("push(7)", X.items)

X.push(6)
print("push(6)", X.items)

X.pop()
print("pop():", X.items)

X.pop()
print("pop():", X.items)

X.push(4)
print("push(4)", X.items)

X.pop()
print("pop():", X.items)

X.pop()
print("pop():", X.items)
```

```
push(5) [5]
push(3) [5, 3]
pop(): [5]
push(2) [5, 2]
push(8) [5, 2, 8]
pop(): [5, 2]
pop(): [5]
push(9) [5, 9]
push(1) [5, 9, 1]
pop(): [5, 9]
push(7) [5, 9, 7]
push(6) [5, 9, 7, 6]
pop(): [5, 9, 7]
pop(): [5, 9]
push(4) [5, 9, 4]
pop(): [5, 9]
pop(): [5]
```