

ACTIVITY GUIDE 1

CODE:

```
3 usages (1 dynamic) new *
class Stack:

    new *
    def __init__(self):
        self.items = []

    new *
    def push(self, item):
        self.items.append(item)

    new *
    def pop(self):
        if self.is_empty():
            print("Error: Stack is empty")
            return None
        return self.items.pop()

    new *
    def top(self):
        if self.is_empty():
            print("Error: Stack is empty")
            return None
        return self.items[-1]

4 usages new *
    def is_empty(self):
        return len(self.items) == 0

    new *
    def __len__(self):
        return len(self.items)
```

```
27 # Part 1: Initial Operations from the Table
28 print("=== Part 1: Initial Operations ===")
29 S = Stack()
30
31 S.push(5)          # Stack: [5]
32 print(f"Operation: S.push(5) -> Stack: {S.items}")
33
34 S.push(3)          # Stack: [5, 3]
35 print(f"Operation: S.push(3) -> Stack: {S.items}")
36
37 print(f"Operation: len(S) -> Output: {len(S)}")
38
39 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
40
41 print(f"Operation: S.is_empty() -> Output: {S.is_empty()}")
42
43 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
44
45 print(f"Operation: S.is_empty() -> Output: {S.is_empty()}")
46
47 print(f"Operation: S.pop() -> Output: {S.pop()} (Error expected) -> Stack: {S.items}")
48
49 S.push(7)          # Stack: [7]
50 print(f"Operation: S.push(7) -> Stack: {S.items}")
51
52 S.push(9)          # Stack: [7, 9]
53 print(f"Operation: S.push(9) -> Stack: {S.items}")
54
55 print(f"Operation: S.top() -> Output: {S.top()}")
56
57 S.push(4)          # Stack: [7, 9, 4]
58 print(f"Operation: S.push(4) -> Stack: {S.items}")
59
60 print(f"Operation: len(S) -> Output: {len(S)}")
61
62 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
63
64 S.push(6)          # Stack: [7, 9, 6]
65 print(f"Operation: S.push(6) -> Stack: {S.items}")
```

```

67 S.push(8)          # Stack: [7, 9, 6, 8]
68 print(f"Operation: S.push(8) -> Stack: {S.items}")
69
70 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
71
72 # Part 2: Additional Operations from the Second Sequence
73 print("\n== Part 2: Additional Operations ==")
74 S = Stack() # Start with a new empty stack
75
76 S.push(5)          # Stack: [5]
77 print(f"Operation: S.push(5) -> Stack: {S.items}")
78
79 S.push(3)          # Stack: [5, 3]
80 print(f"Operation: S.push(3) -> Stack: {S.items}")
81
82 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
83
84 S.push(2)          # Stack: [5, 2]
85 print(f"Operation: S.push(2) -> Stack: {S.items}")
86
87 S.push(8)          # Stack: [5, 2, 8]
88 print(f"Operation: S.push(8) -> Stack: {S.items}")
89
90 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
91
92 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
93
94 S.push(9)          # Stack: [5, 9]
95 print(f"Operation: S.push(9) -> Stack: {S.items}")
96
97 S.push(1)          # Stack: [5, 9, 1]
98 print(f"Operation: S.push(1) -> Stack: {S.items}")
99
100 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
101
102 S.push(7)          # Stack: [5, 9, 7]
103 print(f"Operation: S.push(7) -> Stack: {S.items}")
104
105 S.push(6)          # Stack: [5, 9, 7, 6]
106 print(f"Operation: S.push(6) -> Stack: {S.items}")

```

```

108 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
109
110 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
111
112 S.push(4)
113 print(f"Operation: S.push(4) -> Stack: {S.items}")
114
115 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")
116
117 print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

```

OUTPUT: PART 1

```
=== Part 1: Initial Operations ===
Operation: S.push(5) -> Stack: [5]
Operation: S.push(3) -> Stack: [5, 3]
Operation: len(S) -> Output: 2
Operation: S.pop() -> Output: 3 -> Stack: [5]
Operation: S.is_empty() -> Output: False
Operation: S.pop() -> Output: 5 -> Stack: []
Operation: S.is_empty() -> Output: True
Error: Stack is empty
Operation: S.pop() -> Output: None (Error expected) -> Stack: []
Operation: S.push(7) -> Stack: [7]
```

```
Operation: S.push(9) -> Stack: [7, 9]
Operation: S.top() -> Output: 9
Operation: S.push(4) -> Stack: [7, 9, 4]
Operation: len(S) -> Output: 3
Operation: S.pop() -> Output: 4 -> Stack: [7, 9]
Operation: S.push(6) -> Stack: [7, 9, 6]
Operation: S.push(8) -> Stack: [7, 9, 6, 8]
Operation: S.pop() -> Output: 8 -> Stack: [7, 9, 6]
```

PART 2:

```
=== Part 2: Additional Operations ===
Operation: S.push(5) -> Stack: [5]
Operation: S.push(3) -> Stack: [5, 3]
Operation: S.pop() -> Output: 3 -> Stack: [5]
Operation: S.push(2) -> Stack: [5, 2]
Operation: S.push(8) -> Stack: [5, 2, 8]
Operation: S.pop() -> Output: 8 -> Stack: [5, 2]
Operation: S.pop() -> Output: 2 -> Stack: [5]
Operation: S.push(9) -> Stack: [5, 9]
Operation: S.push(1) -> Stack: [5, 9, 1]
Operation: S.pop() -> Output: 1 -> Stack: [5, 9]
```

```
Operation: S.push(7) -> Stack: [5, 9, 7]
Operation: S.push(6) -> Stack: [5, 9, 7, 6]
Operation: S.pop() -> Output: 6 -> Stack: [5, 9, 7]
Operation: S.pop() -> Output: 7 -> Stack: [5, 9]
Operation: S.push(4) -> Stack: [5, 9, 4]
Operation: S.pop() -> Output: 4 -> Stack: [5, 9]
Operation: S.pop() -> Output: 9 -> Stack: [5]
```