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)
```

```
# Part 1: Initial Operations from the Table
print("=== Part 1: Initial Operations ===")
S = Stack()

S.push(5)              # Stack: [5]
print(f"Operation: S.push(5) -> Stack: {S.items}")

S.push(3)              # Stack: [5, 3]
print(f"Operation: S.push(3) -> Stack: {S.items}")

print(f"Operation: len(S) -> Output: {len(S)}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

print(f"Operation: S.is_empty() -> Output: {S.is_empty()}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

print(f"Operation: S.is_empty() -> Output: {S.is_empty()}")

print(f"Operation: S.pop() -> Output: {S.pop()} (Error expected) -> Stack: {S.items}")

S.push(7)              # Stack: [7]
print(f"Operation: S.push(7) -> Stack: {S.items}")

S.push(9)              # Stack: [7, 9]
print(f"Operation: S.push(9) -> Stack: {S.items}")

print(f"Operation: S.top() -> Output: {S.top()}")

S.push(4)              # Stack: [7, 9, 4]
print(f"Operation: S.push(4) -> Stack: {S.items}")

print(f"Operation: len(S) -> Output: {len(S)}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

S.push(6)              # Stack: [7, 9, 6]
print(f"Operation: S.push(6) -> Stack: {S.items}")
```

```
S.push(8)              # Stack: [7, 9, 6, 8]
print(f"Operation: S.push(8) -> Stack: {S.items}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

# Part 2: Additional Operations from the Second Sequence
print("\n=== Part 2: Additional Operations ===")
S = Stack()  # Start with a new empty stack

S.push(5)              # Stack: [5]
print(f"Operation: S.push(5) -> Stack: {S.items}")

S.push(3)              # Stack: [5, 3]
print(f"Operation: S.push(3) -> Stack: {S.items}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

S.push(2)              # Stack: [5, 2]
print(f"Operation: S.push(2) -> Stack: {S.items}")

S.push(8)              # Stack: [5, 2, 8]
print(f"Operation: S.push(8) -> Stack: {S.items}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

S.push(9)              # Stack: [5, 9]
print(f"Operation: S.push(9) -> Stack: {S.items}")

S.push(1)              # Stack: [5, 9, 1]
print(f"Operation: S.push(1) -> Stack: {S.items}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

S.push(7)              # Stack: [5, 9, 7]
print(f"Operation: S.push(7) -> Stack: {S.items}")

S.push(6)              # Stack: [5, 9, 7, 6]
print(f"Operation: S.push(6) -> Stack: {S.items}")
```

```
print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

S.push(4)
print(f"Operation: S.push(4) -> Stack: {S.items}")

print(f"Operation: S.pop() -> Output: {S.pop()} -> Stack: {S.items}")

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]
```