```
# 1. Stack and Queue
from queue import Queue
print("\n--- 1. Stack and Queue ---")

queue = Queue()
for i in range(1, 6):
    queue.put(i)

stack = []
while not queue.empty():
    stack.append(queue.get())
while stack:
    queue.put(stack.pop())

print("Reversed Queue:", end=" ")
while not queue.empty():
    print(queue.get(), end=" ")

# 2. Stack and Linked List
print("\n\n--- 2. Stack and Linked List ---")

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:
    def __init__(self):
```

```
        self.head = None

    def push(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def pop(self):
        if not self.head:
            return "Stack is empty"
        data = self.head.data
        self.head = self.head.next
        return data

stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
print(stack.pop())  # 3
print(stack.pop())  # 2
print(stack.pop())  # 1

# 3. Stack and Array
print("\n--- 3. Stack and Array ---")

stack = []
stack.append(10)
stack.append(20)
```

```
stack.append(30)

print("Stack Elements:", end=" ")
while stack:
    print(stack.pop(), end=" ")

# 4. Stack and Tree
print("\n\n--- 4. Stack and Tree ---")

class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = self.right = None

def inorder_traversal(root):
    stack, result = [], []
    current = root
    while current or stack:
        while current:
            stack.append(current)
            current = current.left
        current = stack.pop()
        result.append(current.value)
        current = current.right
    return result

root = TreeNode(1)
root.right = TreeNode(2)
```

```
root.right.left = TreeNode(3)

print("In-order Traversal:", inorder_traversal(root))

# 5. Stack and Graph
print("\n--- 5. Stack and Graph ---")

graph = {
  'A': ['B', 'C'],
  'B': ['D', 'E'],
  'C': ['F'],
  'D': [],
  'E': [],
  'F': []
}

def dfs_iterative(graph, start):
  stack, visited = [start], []
  while stack:
    node = stack.pop()
    if node not in visited:
      visited.append(node)
      stack.extend(graph[node][::-1])  # Reverse to maintain order
  return visited

print("DFS Order:", dfs_iterative(graph, 'A'))

# 6. Stack and Hash Map
```

```
print("\n--- 6. Stack and Hash Map ---")

nums = [2, 1, 2, 4, 3]
stack, result = [], {}
for num in nums:
    while stack and stack[-1] < num:
        result[stack.pop()] = num
    stack.append(num)
while stack:
    result[stack.pop()] = -1

print("Nearest Greater Elements:", [result[num] for num in nums])

# 7. Stack and Heap
print("\n--- 7. Stack and Heap ---")

import heapq
stack = []
heap = []

stack.append(("write", 1))
stack.append(("edit", 2))
heapq.heappush(heap, (1, "write"))
heapq.heappush(heap, (2, "edit"))

print("Undoing last operation:", stack.pop())
priority, action = heapq.heappop(heap)
print("Processing priority operation:", action)
```

```
# 8. Stack and Matrix
print("\n--- 8. Stack and Matrix ---")

def largest_rectangle(heights):
    stack, max_area = [], 0
    heights.append(0)  # Add sentinel for easier computation
    for i, h in enumerate(heights):
        while stack and heights[stack[-1]] > h:
            height = heights[stack.pop()]
            width = i if not stack else i - stack[-1] - 1
            max_area = max(max_area, height * width)
        stack.append(i)
    return max_area

heights = [2, 1, 5, 6, 2, 3]
print("Largest Rectangle Area:", largest_rectangle(heights))
```