```python
#1. Reverse a String Using a Stack
def reverse_string(s):
    stack = []
    for char in s:
        stack.append(char)
    reversed_str = ""
    while stack:
        reversed_str += stack.pop()
    return reversed_str
string = "hello"
print("Original:", string)
print("Reversed:", reverse_string(string))
```

```
Original: hello
Reversed: olleh
```

```python
#2. Stack Sort Algorithm
def sort_stack(stack):
    temp_stack = []
    while stack:
        temp = stack.pop()
        while temp_stack and temp_stack[-1] > temp:
            stack.append(temp_stack.pop())
        temp_stack.append(temp)
    return temp_stack
stack = [34, 3, 31, 98, 92, 23]
print("Original Stack:", stack)
sorted_stack = sort_stack(stack)
print("Sorted Stack:", sorted_stack)
```

```
Original Stack: [34, 3, 31, 98, 92, 23]
Sorted Stack: [3, 23, 31, 34, 92, 98]
```

```python
#3. Implement a Stack for Queue Operations
class QueueUsingStacks:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def enqueue(self, x):
        self.stack1.append(x)

    def dequeue(self):
        if not self.stack2:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        return self.stack2.pop() if self.stack2 else "Queue is empty"
q = QueueUsingStacks()
q.enqueue(1)
q.enqueue(2)
```

```
q.enqueue(3)
print(q.dequeue())
print(q.dequeue())
```

```
1
2
```

```
#4. Undo/Redo Operation with Stack
class UndoRedo:
    def __init__(self):
        self.undo_stack = []
        self.redo_stack = []

    def perform_action(self, action):
        self.undo_stack.append(action)
        self.redo_stack.clear()

    def undo(self):
        if self.undo_stack:
            action = self.undo_stack.pop()
            self.redo_stack.append(action)
            return action
        return "Nothing to undo"

    def redo(self):
        if self.redo_stack:
            action = self.redo_stack.pop()
            self.undo_stack.append(action)
            return action
        return "Nothing to redo"
editor = UndoRedo()
editor.perform_action("Type A")
editor.perform_action("Type B")
print(editor.undo())
print(editor.redo())
```

```
Type B
Type B
```

```
#5. Stack That Supports Getting the Middle Element
class StackWithMiddle:
    def __init__(self):
        self.stack = []

    def push(self, x):
        self.stack.append(x)

    def pop(self):
        return self.stack.pop() if self.stack else "Stack is empty"

    def get_middle(self):
```

```
        if self.stack:
            return self.stack[len(self.stack) // 2]
        return "Stack is empty"
stack = StackWithMiddle()
stack.push(1)
stack.push(2)
stack.push(3)
print(stack.get_middle())
stack.push(4)
print(stack.get_middle())
```

```
2
3
```

```
#6. Remove Duplicates from a Stack
def remove_duplicates(stack):
    seen = set()
    unique_stack = []
    for elem in stack:
        if elem not in seen:
            seen.add(elem)
            unique_stack.append(elem)
    return unique_stack
stack = [1, 2, 3, 1, 2, 4]
print("Original Stack:", stack)
print("After Removing Duplicates:", remove_duplicates(stack))
```

```
Original Stack: [1, 2, 3, 1, 2, 4]
After Removing Duplicates: [1, 2, 3, 4]
```

```
#8. Check if a Stack is Palindrome
def is_palindrome(stack):
    return stack == stack[::-1]
stack = [1, 2, 3, 2, 1]
print("Is Palindrome?", is_palindrome(stack))
stack = [1, 2, 3]
print("Is Palindrome?", is_palindrome(stack))
```

```
Is Palindrome? True
Is Palindrome? False
```

```
#9. Next Greater Element Using Stack
def next_greater_element(nums):
    stack = []
    result = [-1] * len(nums)
    for i in range(len(nums) - 1, -1, -1):
        while stack and stack[-1] <= nums[i]:
            stack.pop()
        if stack:
            result[i] = stack[-1]
        stack.append(nums[i])
```

```
    return result
nums = [4, 5, 2, 10]
print("Next Greater Elements:", next_greater_element(nums))
```

```
Next Greater Elements: [5, 10, 10, -1]
```

```
#10. Reverse a Queue Using a Stack
from collections import deque

def reverse_queue(queue):
    stack = []
    while queue:
        stack.append(queue.popleft())
    while stack:
        queue.append(stack.pop())
    return queue

queue = deque([1, 2, 3, 4])
print("Original Queue:", list(queue))
reversed_queue = reverse_queue(queue)
print("Reversed Queue:", list(reversed_queue))
```

```
Original Queue: [1, 2, 3, 4]
Reversed Queue: [4, 3, 2, 1]
```

```
#11. Remove All Elements Less Than X
def remove_less_than_x(stack, x):
    return [elem for elem in stack if elem >= x]
stack = [1, 5, 3, 8, 6]
print("Original Stack:", stack)
modified_stack = remove_less_than_x(stack, 5)
print("Modified Stack:", modified_stack)
```

```
Original Stack: [1, 5, 3, 8, 6]
Modified Stack: [5, 8, 6]
```