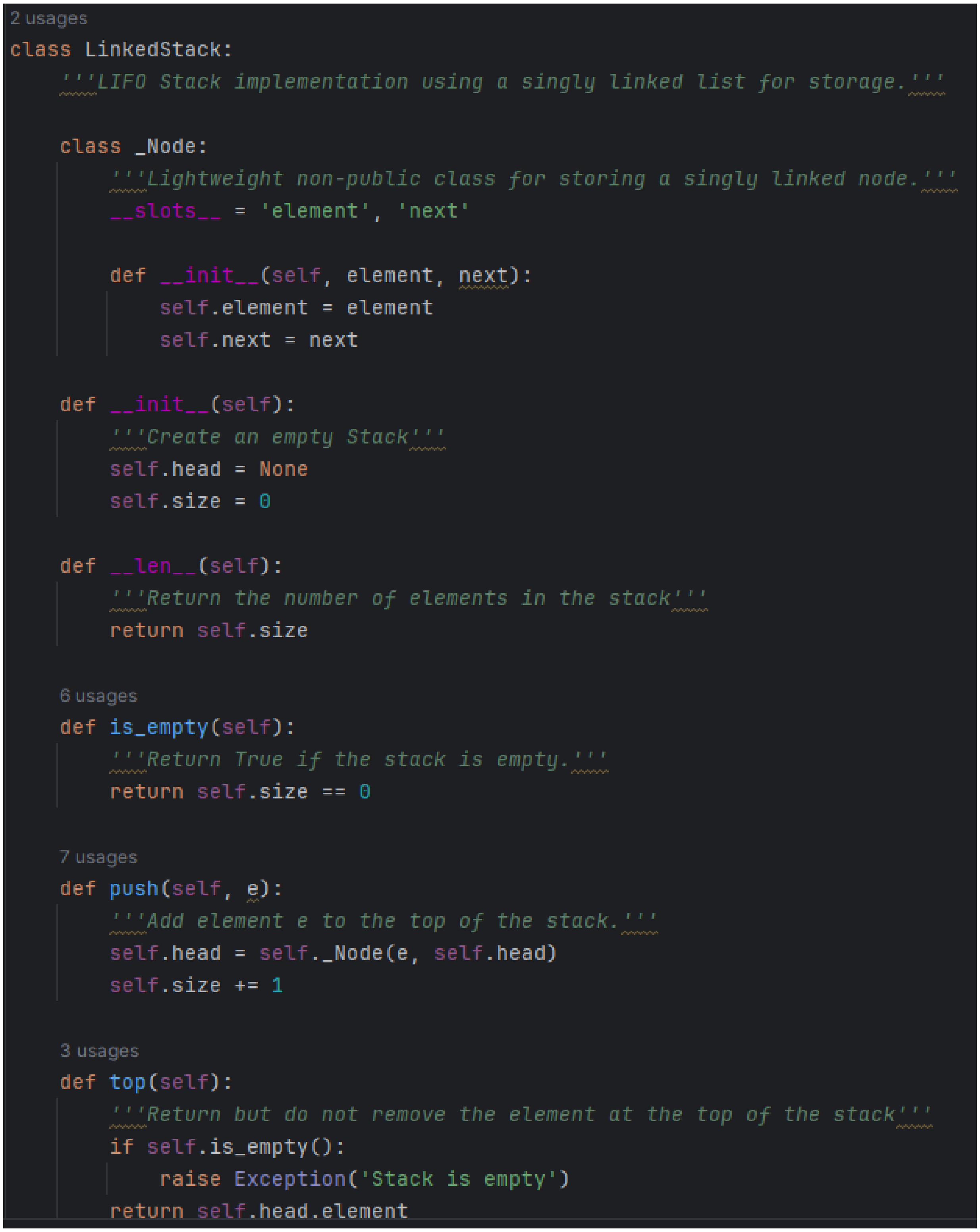
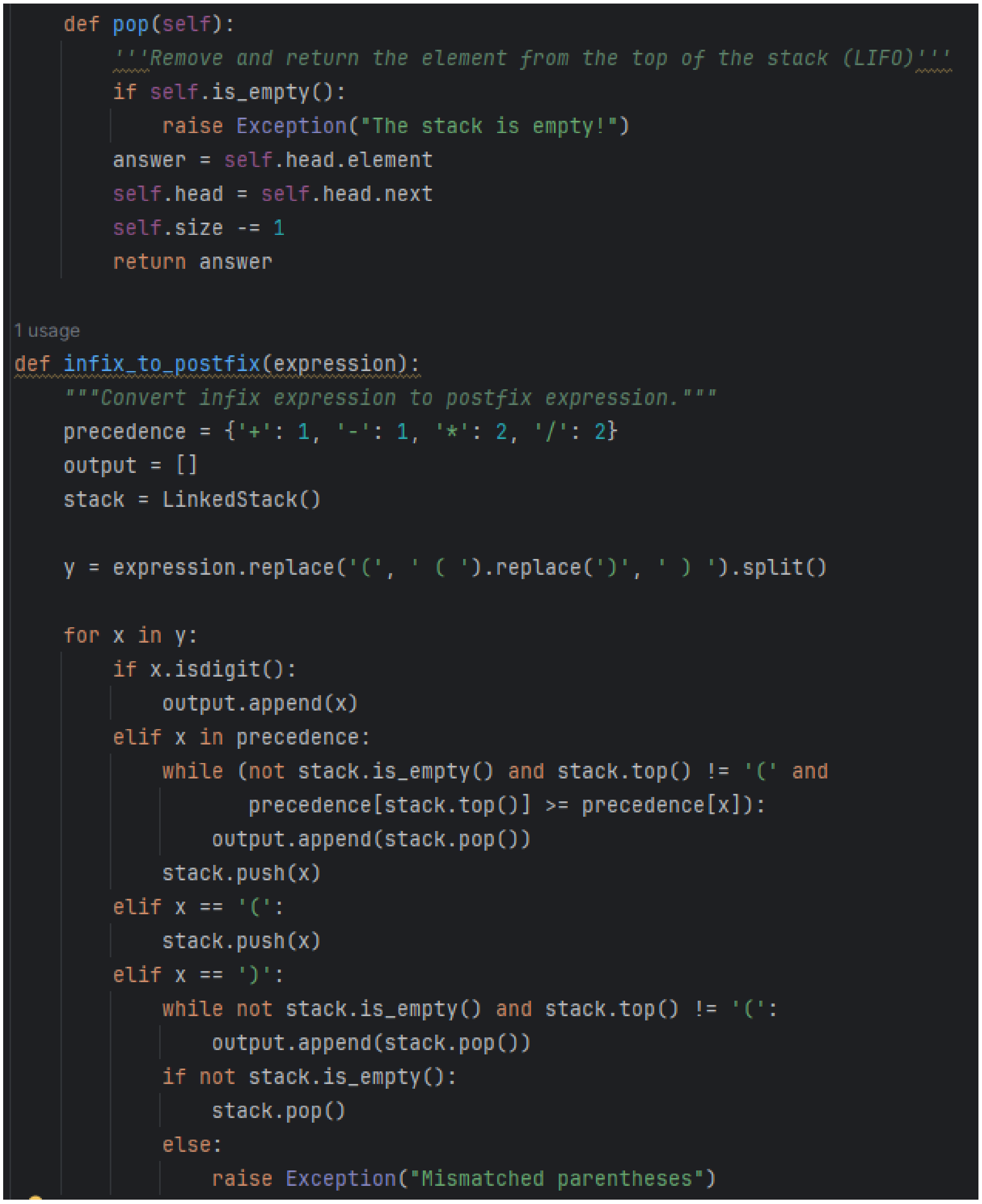
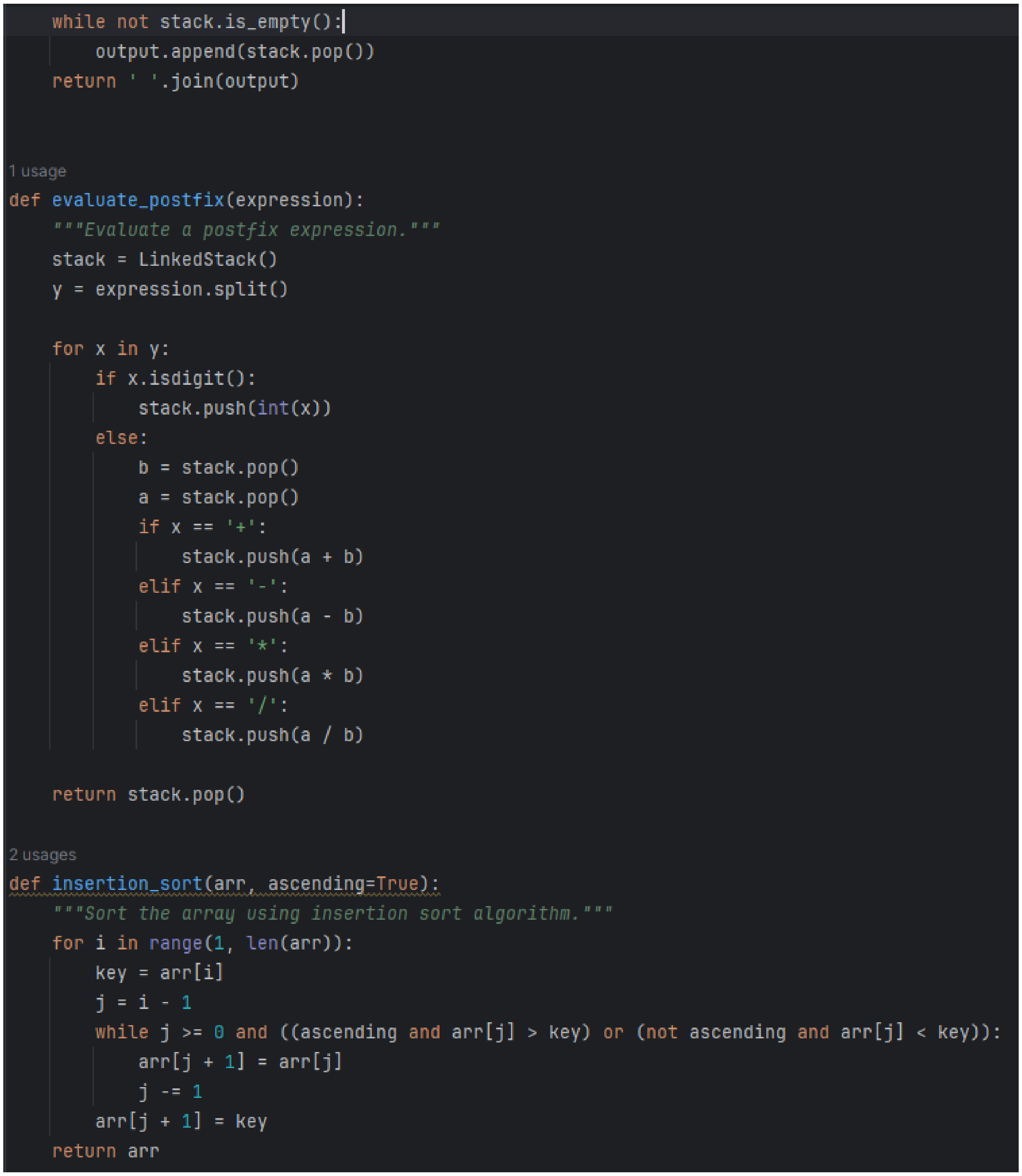
Code:



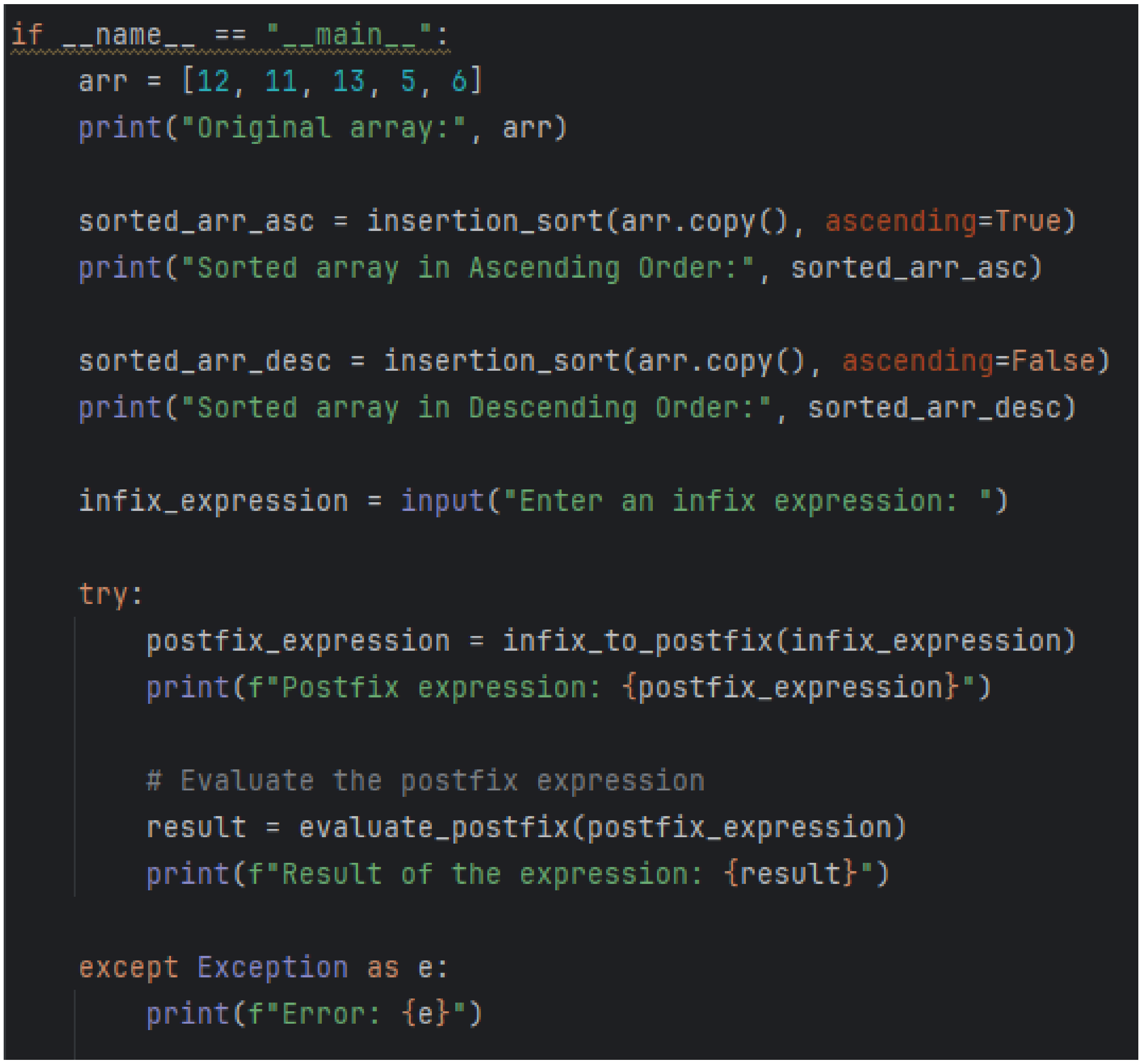
```
2 usages
class LinkedStack:
    '''LIFO Stack implementation using a singly linked list for storage.'''

    class _Node:
        '''Lightweight non-public class for storing a singly linked node.'''
        __slots__ = 'element', 'next'

        def __init__(self, element, next):
            self.element = element
            self.next = next

    def __init__(self):
        '''Create an empty Stack'''
        self.head = None
        self.size = 0

    def __len__(self):
        '''Return the number of elements in the stack'''
        return self.size

    6 usages
    def is_empty(self):
        '''Return True if the stack is empty.'''
        return self.size == 0

    7 usages
    def push(self, e):
        '''Add element e to the top of the stack.'''
        self.head = self._Node(e, self.head)
        self.size += 1

    3 usages
    def top(self):
        '''Return but do not remove the element at the top of the stack'''
        if self.is_empty():
            raise Exception('Stack is empty')
        return self.head.element
```



```python
    def pop(self):
        '''Remove and return the element from the top of the stack (LIFO)'''
        if self.is_empty():
            raise Exception("The stack is empty!")
        answer = self.head.element
        self.head = self.head.next
        self.size -= 1
        return answer

1 usage
def infix_to_postfix(expression):
    """Convert infix expression to postfix expression."""
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2}
    output = []
    stack = LinkedStack()

    y = expression.replace('(', ' ( ').replace(')', ' ) ').split()

    for x in y:
        if x.isdigit():
            output.append(x)
        elif x in precedence:
            while (not stack.is_empty() and stack.top() != '(' and
                    precedence[stack.top()] >= precedence[x]):
                output.append(stack.pop())
            stack.push(x)
        elif x == '(':
            stack.push(x)
        elif x == ')':
            while not stack.is_empty() and stack.top() != '(':
                output.append(stack.pop())
            if not stack.is_empty():
                stack.pop()
            else:
                raise Exception("Mismatched parentheses")
```

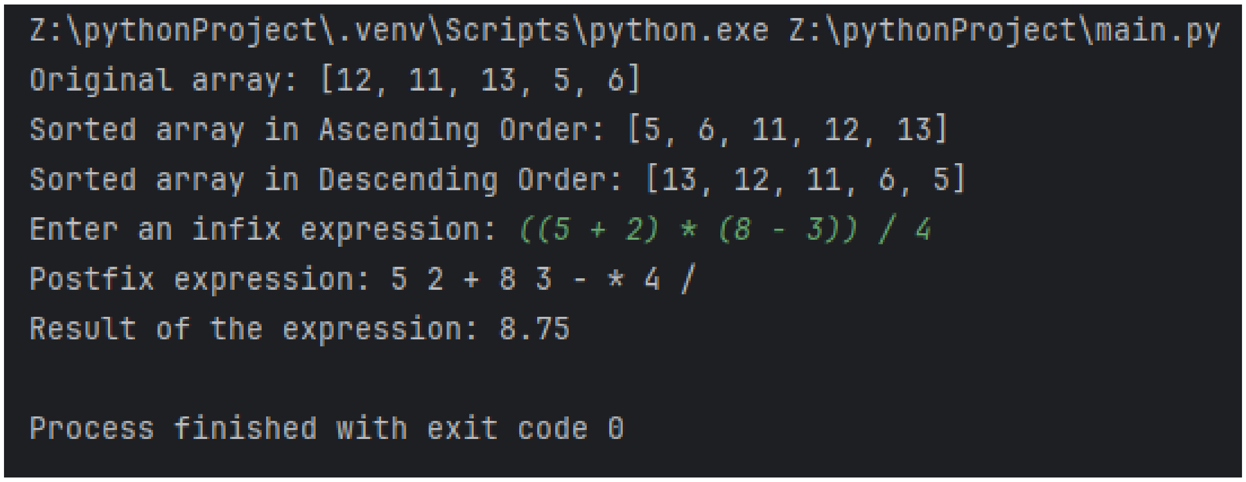


```python
    while not stack.is_empty():
        output.append(stack.pop())
    return ' '.join(output)


1 usage
def evaluate_postfix(expression):
    """Evaluate a postfix expression."""
    stack = LinkedStack()
    y = expression.split()

    for x in y:
        if x.isdigit():
            stack.push(int(x))
        else:
            b = stack.pop()
            a = stack.pop()
            if x == '+':
                stack.push(a + b)
            elif x == '-':
                stack.push(a - b)
            elif x == '*':
                stack.push(a * b)
            elif x == '/':
                stack.push(a / b)

    return stack.pop()

2 usages
def insertion_sort(arr, ascending=True):
    """Sort the array using insertion sort algorithm."""
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and ((ascending and arr[j] > key) or (not ascending and arr[j] < key)):
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
```



```
if __name__ == "__main__":
    arr = [12, 11, 13, 5, 6]
    print("Original array:", arr)

    sorted_arr_asc = insertion_sort(arr.copy(), ascending=True)
    print("Sorted array in Ascending Order:", sorted_arr_asc)

    sorted_arr_desc = insertion_sort(arr.copy(), ascending=False)
    print("Sorted array in Descending Order:", sorted_arr_desc)

    infix_expression = input("Enter an infix expression: ")

    try:
        postfix_expression = infix_to_postfix(infix_expression)
        print(f"Postfix expression: {postfix_expression}")

        # Evaluate the postfix expression
        result = evaluate_postfix(postfix_expression)
        print(f"Result of the expression: {result}")

    except Exception as e:
        print(f"Error: {e}")
```

Output:



```
Z:\pythonProject\.venv\Scripts\python.exe Z:\pythonProject\main.py
Original array: [12, 11, 13, 5, 6]
Sorted array in Ascending Order: [5, 6, 11, 12, 13]
Sorted array in Descending Order: [13, 12, 11, 6, 5]
Enter an infix expression: ((5 + 2) * (8 - 3)) / 4
Postfix expression: 5 2 + 8 3 - * 4 /
Result of the expression: 8.75


Process finished with exit code 0
```