The given nested-if solution is logically correct, but it is not a particularly good one. It works for three numbers, yet the heavy nesting makes the logic harder to follow, and extending it becomes painful. Every time a new number is added, the indentation increases and the number of branches grows rapidly. Although the program runs correctly, the structure is inefficient from a readability and maintainability standpoint.

A clearer alternative is to avoid nesting and handle the highest and lowest comparisons separately. This keeps the control flow flat and easier to understand:

```
num1 = int(input('Number 1: '))
num2 = int(input('Number 2: '))
num3 = int(input('Number 3: '))

if num1 >= num2 and num1 >= num3:
    highest = num1
elif num2 >= num1 and num2 >= num3:
    highest = num2
else:
    highest = num3

if num1 <= num2 and num1 <= num3:
    lowest = num1
elif num2 <= num1 and num2 <= num3:
    lowest = num2
else:
    lowest = num3

print("Highest: " + str(highest))
print("Lowest: " + str(lowest))
```

This approach uses more conditions but eliminates deep nesting. The logic is clearer: first find the maximum, then the minimum. It separates the two tasks instead of mixing them inside nested branches. It is easier for a human to read and adjust.

If the task expands to four or five numbers, this flat structure still scales better than the nested version, though both become cumbersome because loops and functions are forbidden. To compare four numbers, each candidate must be checked against the other three using conditions like "if num1 >= num2 and num1 >= num3 and num1 >= num4". For five numbers, each branch expands to four comparisons. The number of conditions grows quickly, and the code becomes long and repetitive, but the flat structure remains more manageable than deep nesting. The overall escalation shows why real-world code would never handle this manually without loops or data structures.