**Problem 1**

1. Tail recursion is a recursive technique where the recursive call is the final step in a function so that no computation takes place after the call. Many compilers can optimize a tail recursive call and turn an actual, recursive process into an iterative one to save space in the call stack. On the other hand, non-tail recursion performs additional operations after the recursive call] which hinders their optimization for tail calls. This usually leads to increased use of memory, simply because each of the function’s calls has to be preserved in the call stack.
2. In most recursive programming, Helper functions are used to handle other parameters /variables which need to be passed to each recursive call as they do not form part of the interface for the main function. They enable separation of the recursive logic from the top-level interface; hence the code becomes more cleaner and sometimes more optimized to performing extra work besides the actual recursive work (like in the case of accumulation in tail recursion).

**Problem 2**

1. **Python code**

def binaryToDecimal(binaryString):

The main function starts the helper with initial bounds

return binaryToDecimalHelper(binaryString, 0, len(binaryString) - 1)

def binaryToDecimalHelper(binaryString, low, high):

Base case: when low surpasses high, stop recursion

if high < low:

return 0

Get the value at the current position

current\_digit = int(binaryString[low])

Recursive call for the next digit

return (current\_digit \* (2 (high - low))) + binaryToDecimalHelper(binaryString, low + 1, high)

**Test Program**

binaryString = input("Enter a binary number: ")

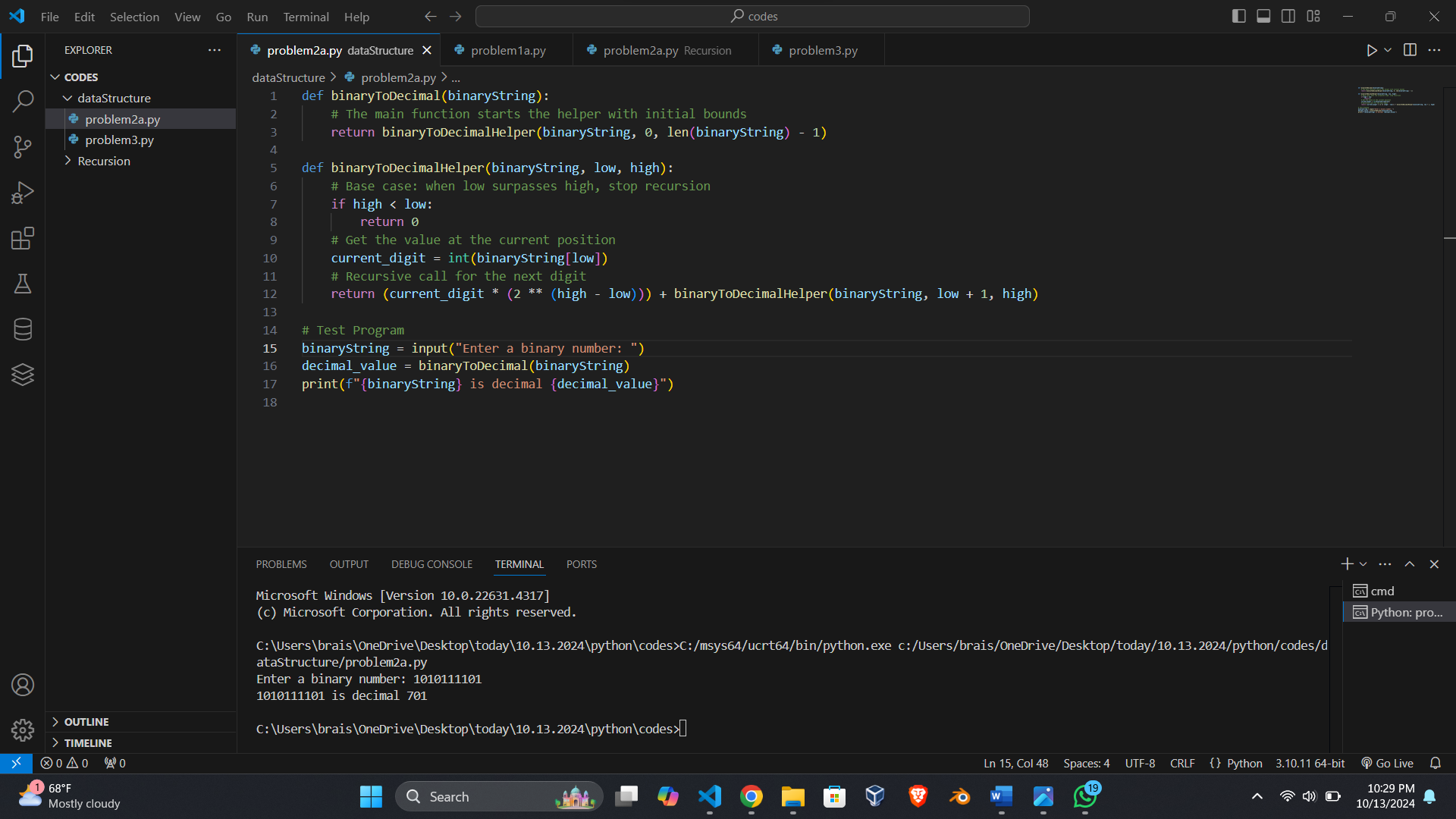
decimal\_value = binaryToDecimal(binaryString)

print(f"{binaryString} is decimal {decimal\_value}")

**Expected Output:**

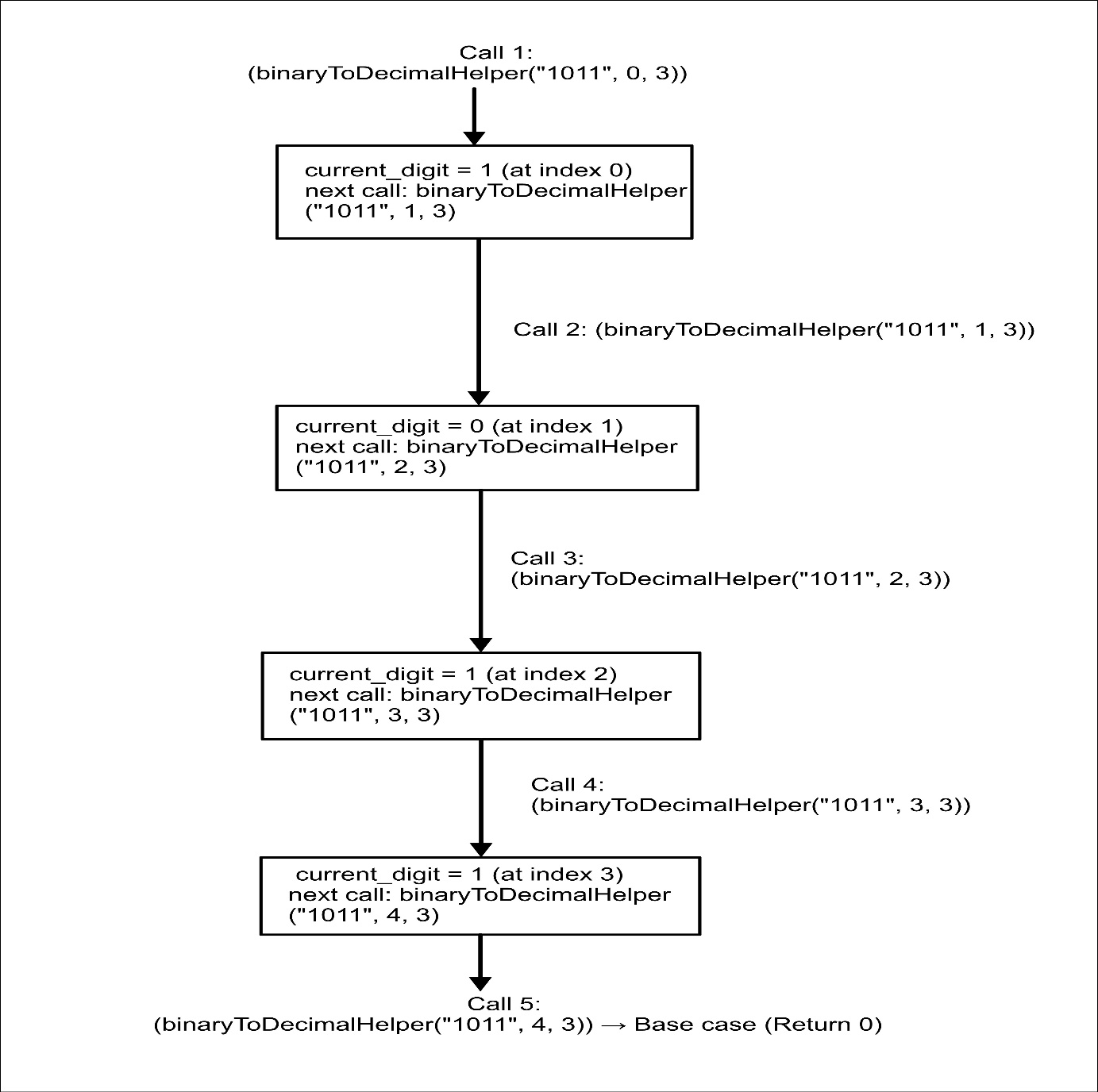
Enter a binary number: 1010111101

1010111101 is decimal 701



1. No, the recursive function is not tail-recursive. The reason is that the recursive call is followed by a computation in which the result from the recursive call is added to `current\_digit \* (2 (high - low))`. Therefore, because there are pending operations after the recursive call, this disqualifies it to be a tail recursion.
2. Sequence of diagrams.

Example, provided the input binary string “1011”

****

**Problem 3**

**Python code:**

def displayPermutation(s):

displayPermutationHelper("", s)

def displayPermutationHelper(s1, s2):

if len(s2) == 0:

print(s1)

else:

for i in range(len(s2)):

displayPermutationHelper(s1 + s2[i], s2[:i] + s2[i+1:])

**Expected Output:**

displayPermutation("abc")

**Expected Output:**

abc

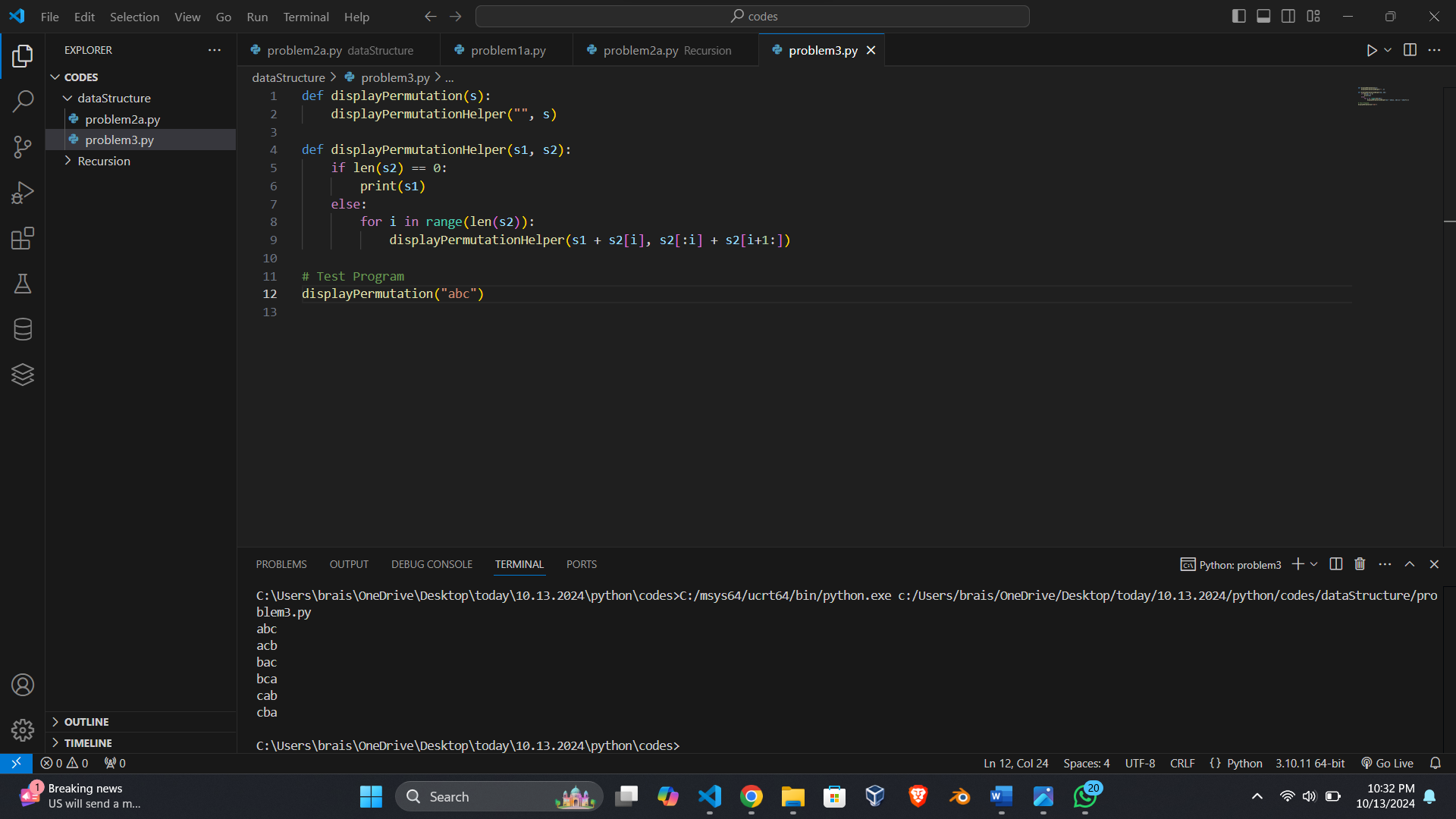
acb

bac

bca

cab

cba



**Test Cases:**

Input: "xyz"

Output:

xyz

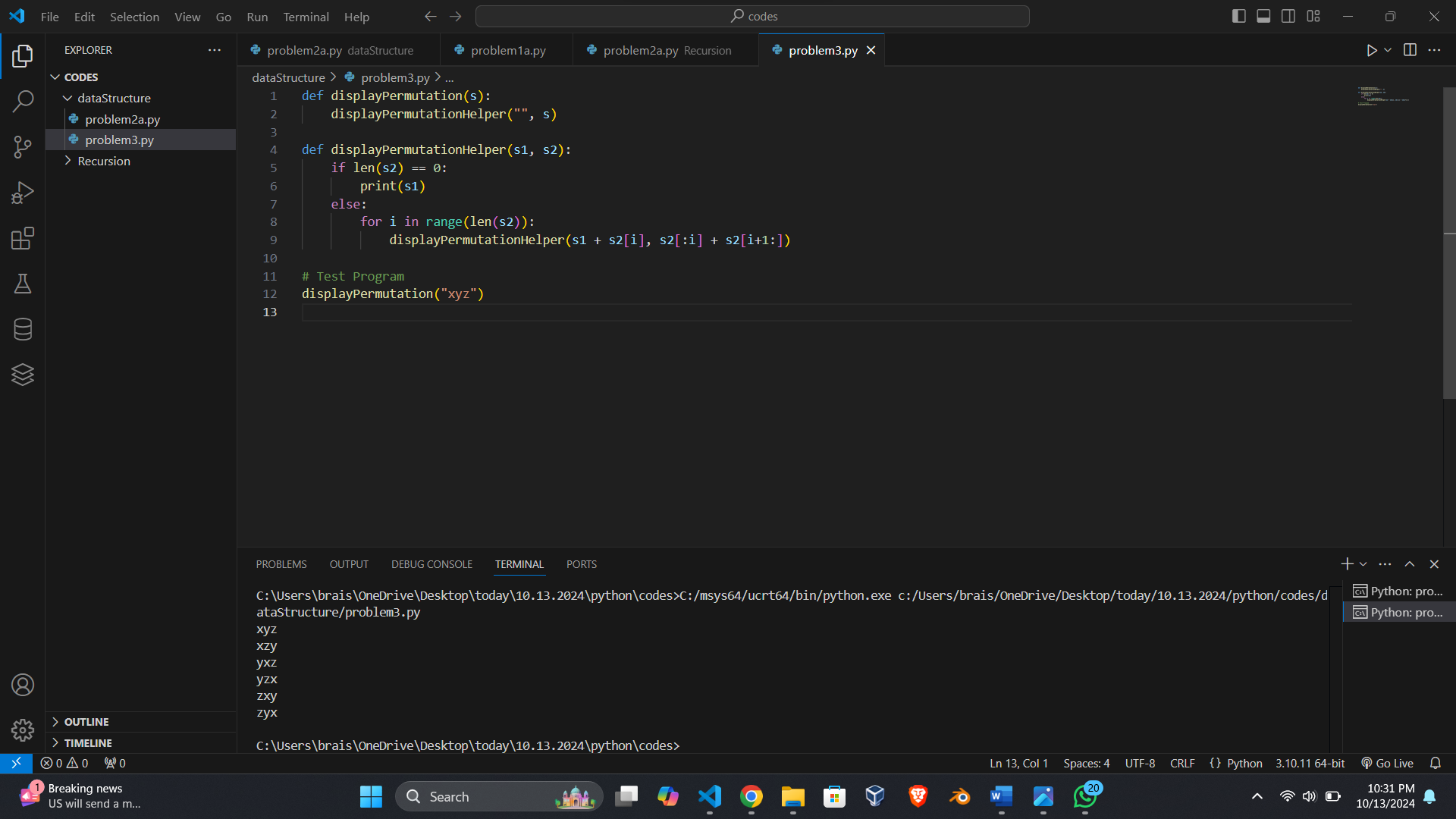
xzy

yxz

yzx

zxy

zyx



**Referred:**

No external sources were used for this solution.