**CPS501: Advanced Programming and Data Structures**

**Problem 1**

1. **Python code**

def is\_substring(s1, s2):

Traverse the first string

for i in range(len(s1) - len(s2) + 1):

Check if the substring from index i matches s2

match = True

for j in range(len(s2)):

if s1[i + j] != s2[j]:

match = False

break

if match:

return i Return the index of the first match

return -1 Return -1 if no match is found

Test program

s1 = input("Enter a string s1: ")

s2 = input("Enter a string s2: ")

index = is\_substring(s1, s2)

if index != -1:

print(f"matched at index { index }")

else:

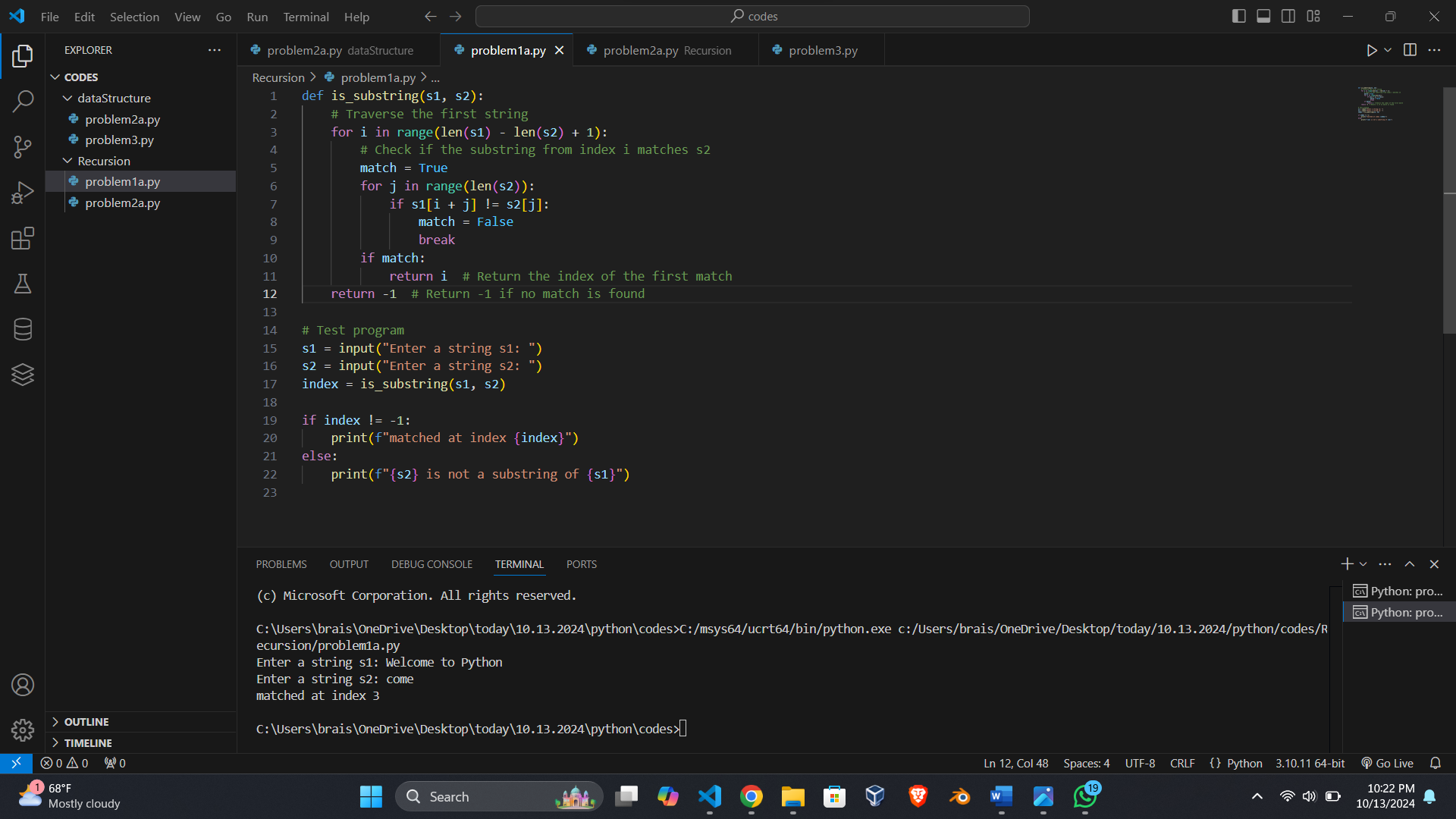
print(f"{s2} is not a substring of { s1 }")

**Expected Output:**

Enter a string s1: **Welcome to Python**

Enter a string s2: **come**

**matched at index 3**

****

1. **time complexity of the algorithm.**

The algorithm has two nested loops:

The outer loop goes through the first string (`s1`) with the index from `0` through the length of the `s1` minus length of the second string `s2` plus one.

The inner loop is to compares one character in `s2` with one corresponding character in `s1`.

The outer loop is the worst case in which the loop runs at most `O(n)` where `n` is the size of the string `s1`. The inner loop runs at most `O(m)` times where `m` is the length of the string `s2`. As a result, the overall time complexity will be equal to n times m, or O(n \* m).

**Problem 2**

1. **Python code**

def longest\_same\_number\_subsequence(numbers):

max\_count = 1

current\_count = 1

longest\_num = numbers[0]

start\_index = 0

longest\_start\_index = 0

for i in range(1, len(numbers)):

if numbers[i] == numbers[i - 1]:

current\_count += 1

else:

if current\_count > max\_count:

max\_count = current\_count

longest\_num = numbers[i - 1]

longest\_start\_index = start\_index

current\_count = 1

start\_index = i

#Check if the last subsequence is the longest

if current\_count > max\_count:

max\_count = current\_count

longest\_num = numbers[-1]

longest\_start\_index = start\_index

return longest\_start\_index, longest\_num, max\_count

#Test program

numbers = list(map(int, input("Enter a series of numbers: ").split()))

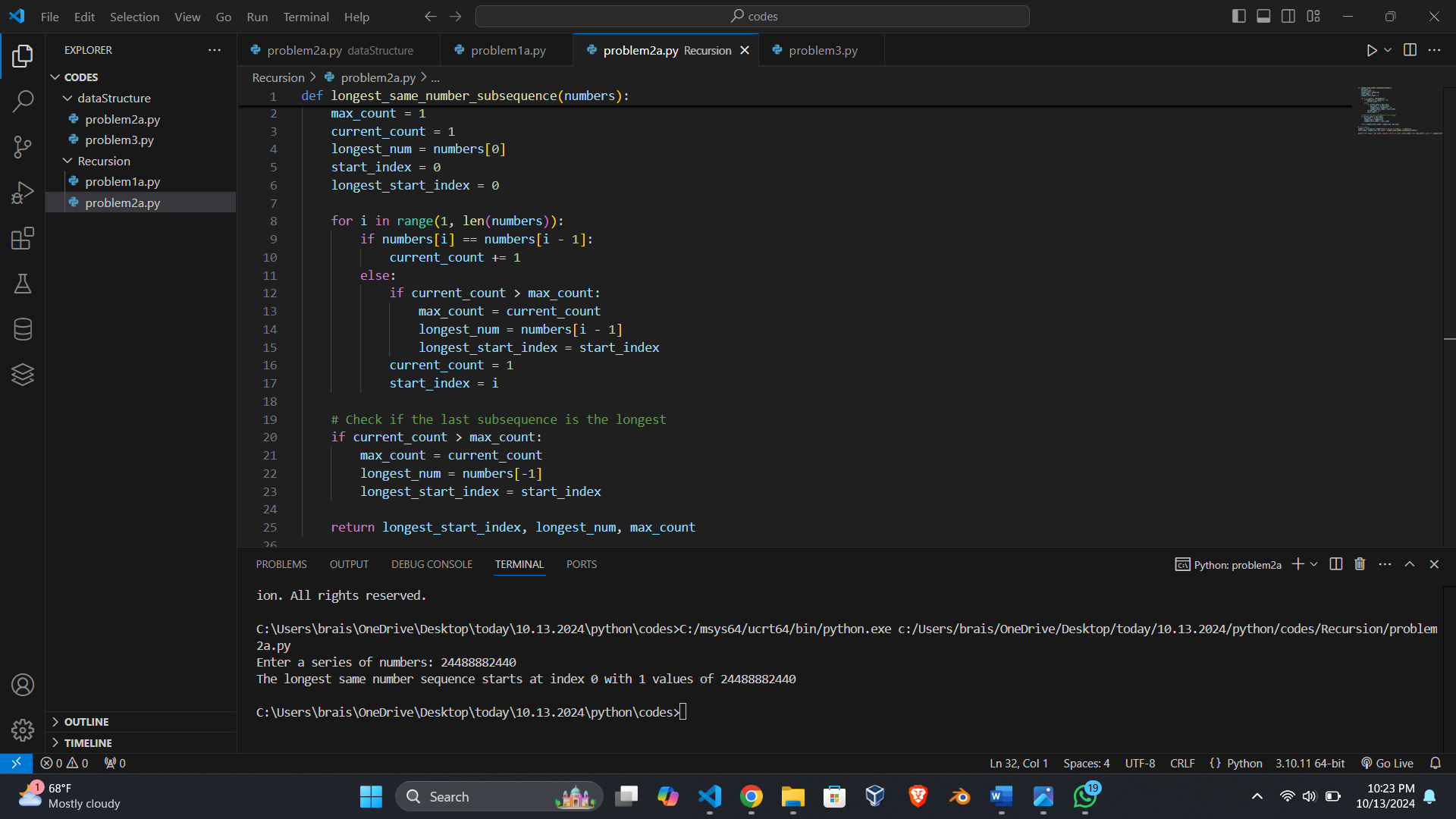
start\_index, longest\_num, max\_count = longest\_same\_number\_subsequence(numbers)

print(f"The longest same number sequence starts at index {start\_index} with {max\_count} values of {longest\_num}")

**Expected Output:**

Enter a series of numbers: 24488882440

The longest same number sequence starts at index 3 with 4 values of 8



1. **Analyze the time complexity of your algorithm.**

This algorithm processes the list of numbers in a single pass and uses a constant amount of additional space to track the longest increasing sequence. Therefore, the time complexity is O(n) whereby n represents the number of integers in the input sequence.

**Test Cases:**

**Test Case 1 for Problem 1:**

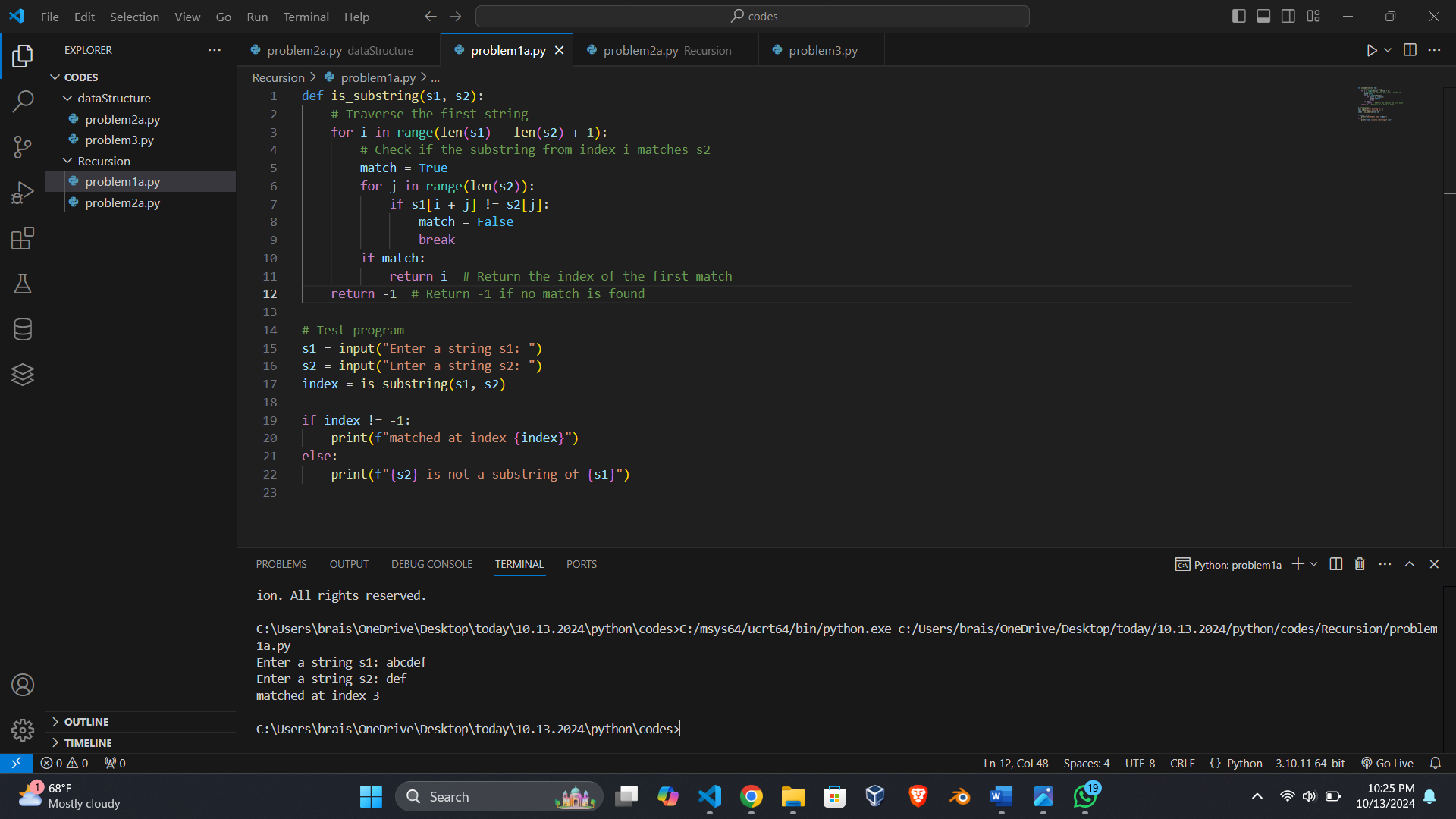
Input:

s1 = “abcdef”

s2 = "def"

Output:

matched at index 3



**Test Case 2 for Problem 1:**

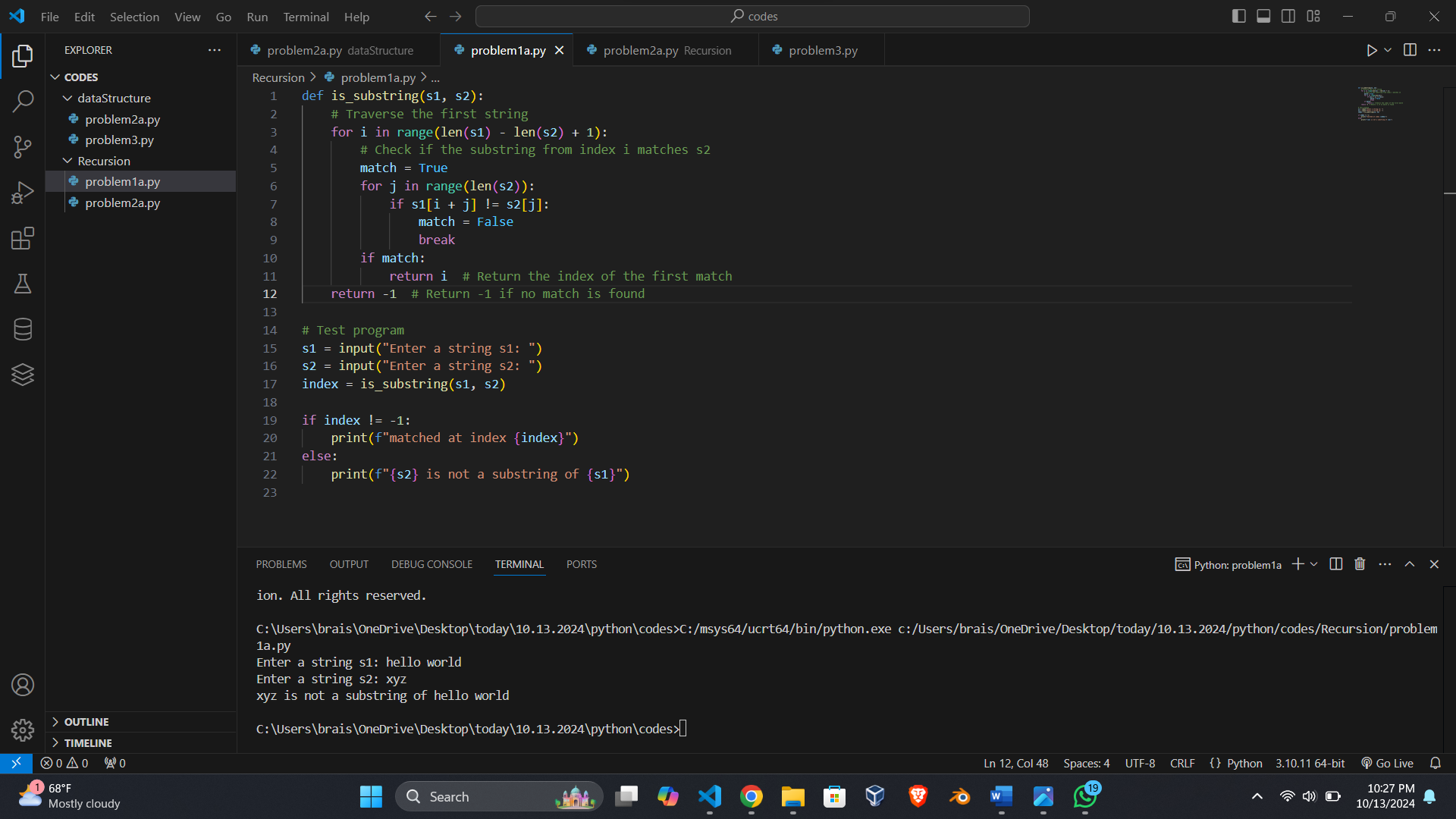
Input:

s1 = "hello world"

s2 = "xyz"

Output:

xyz is not a substring of hello world

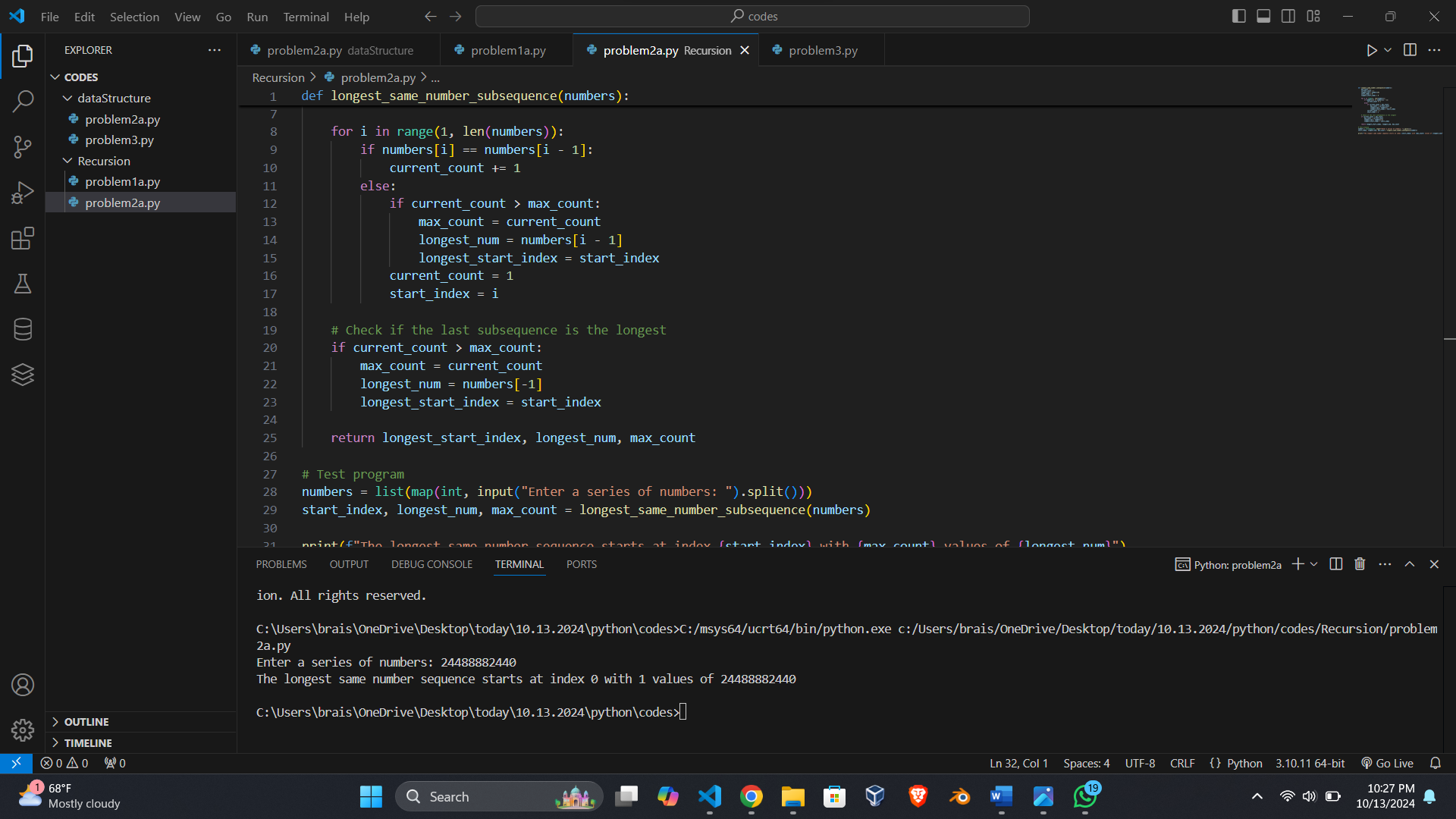


**Test Case 1 for Problem 2:**

Input: 24488882440

Output:

The longest same number sequence starts at index 3 with 4 values of 8

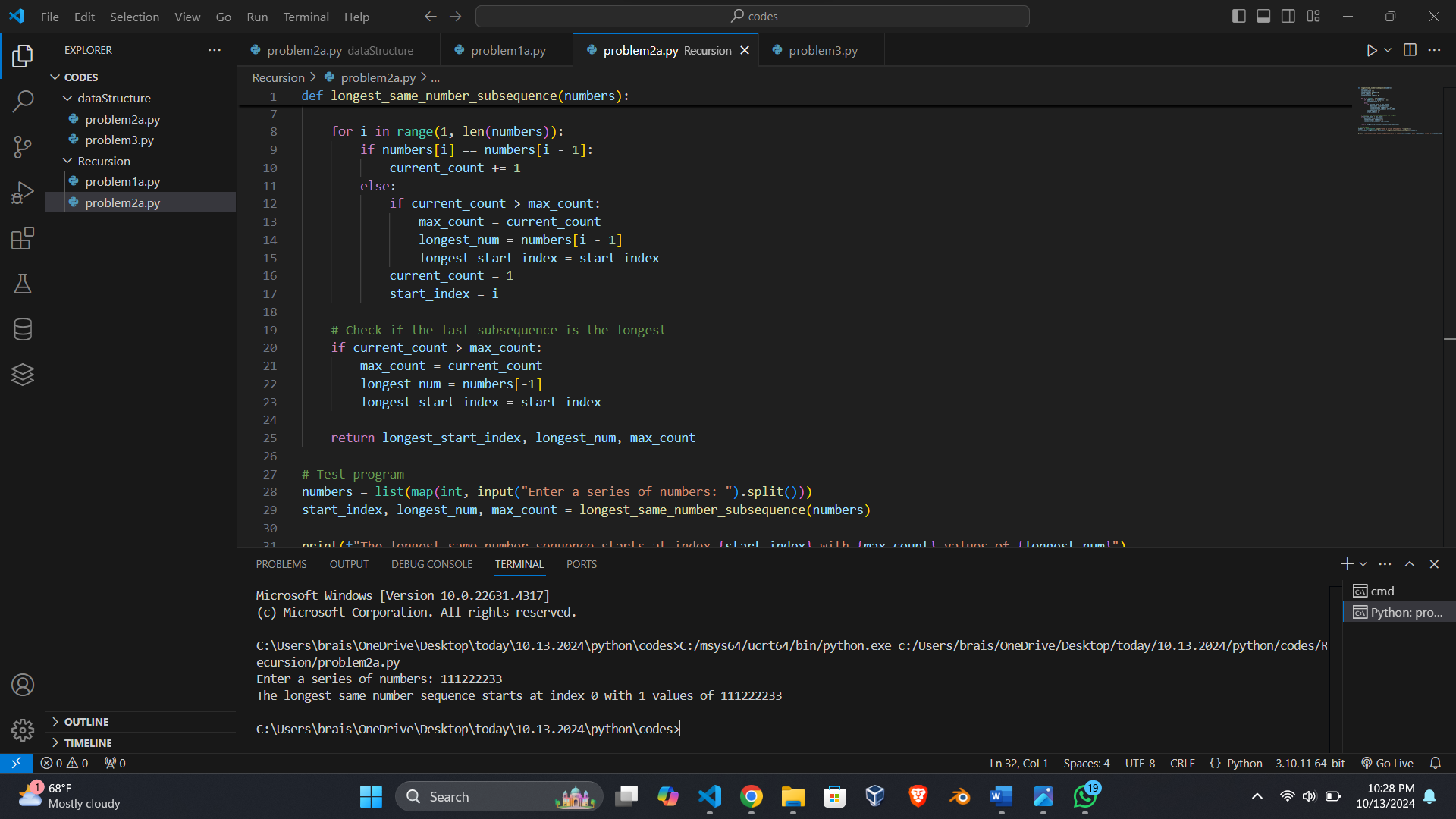


**Test Case 2 for Problem 2:**

Input: 111222233

Output:

The longest same number sequence starts at index 3 with 4 values of 2



**Referred:**

No external sources were used for these solutions.