Python Task-2 Internship Report

**Internship Organization: Main Flow Services and Technologies Pvt. Ltd.**

**Intern Name: Yogesh Patel**

**Task Title: Python Developer Task - 2**

# 1. Objective of the Task

The primary objective of Task-2 was to enhance my understanding of core Python programming by solving a variety of algorithmic and real-world problems. The problems ranged from basic operations like checking for prime numbers to implementing complex logic like maze generation and solving.

**2. Why Learn These Concepts?**

Each problem in this task set reflects a foundational skill in programming:

* Prime Numbers, GCD, and LCM help build strong mathematical logic.
* List Reversal and Sorting improve understanding of data structures.
* String Manipulation and Length Counting strengthen control flow and character-level handling.
* Maze Solving with DFS introduces concepts of recursion, backtracking, and graph traversal.

# 3. Task Breakdown and Explanation

**9. Prime Number Check**: Checks if a number is prime using trial division up to the square root.

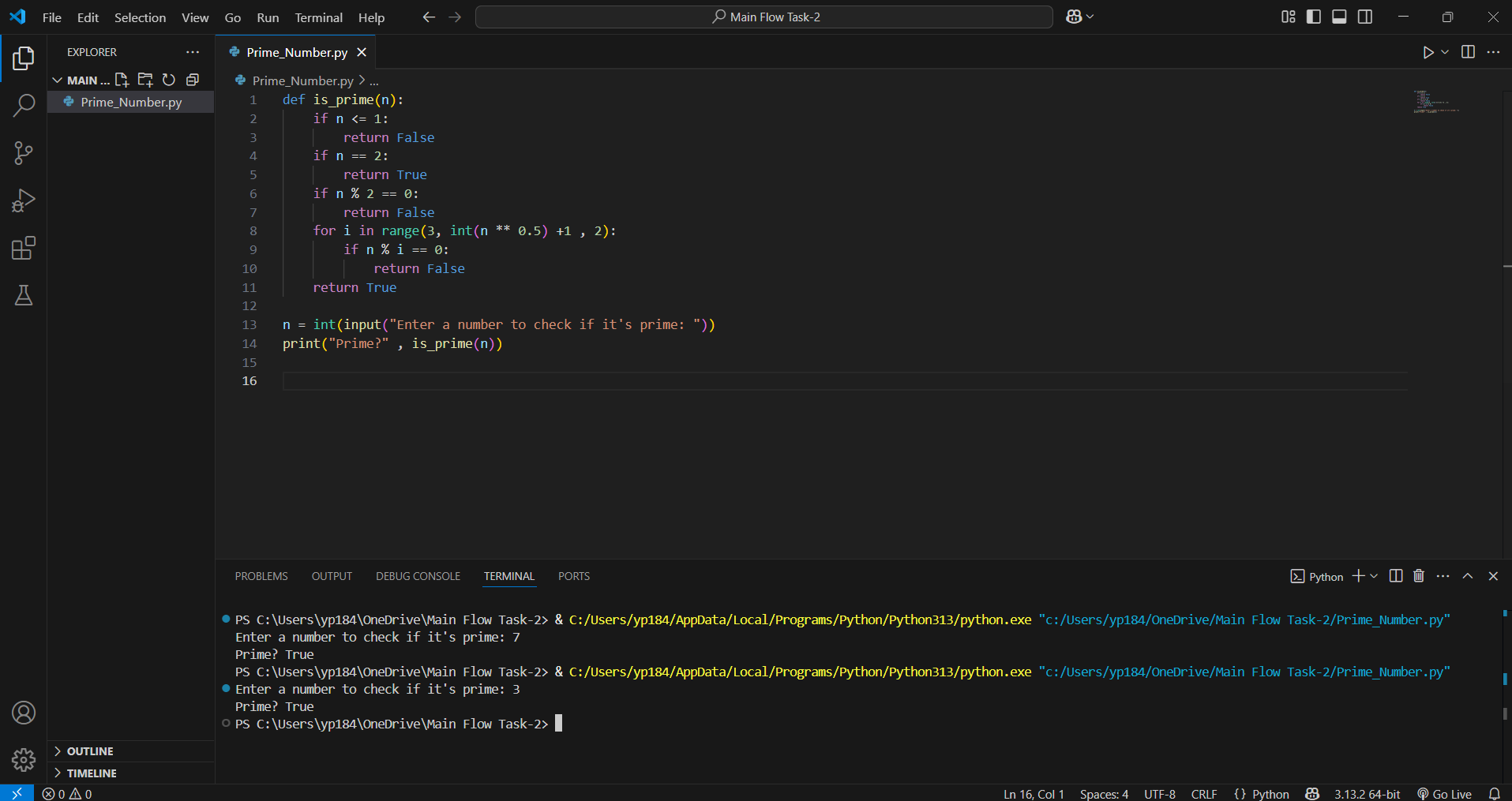
• Code : def is\_prime(n): if n <= 1:

return False if n == 2:

return True if n % 2 == 0: return False for i in range(3, int(n \*\* 0.5) + 1, 2):

if n % i == 0: return False return True

n = int(input("Enter a number to check if it's prime: ")) print("Prime?" , is\_prime(n))

screenshot: 

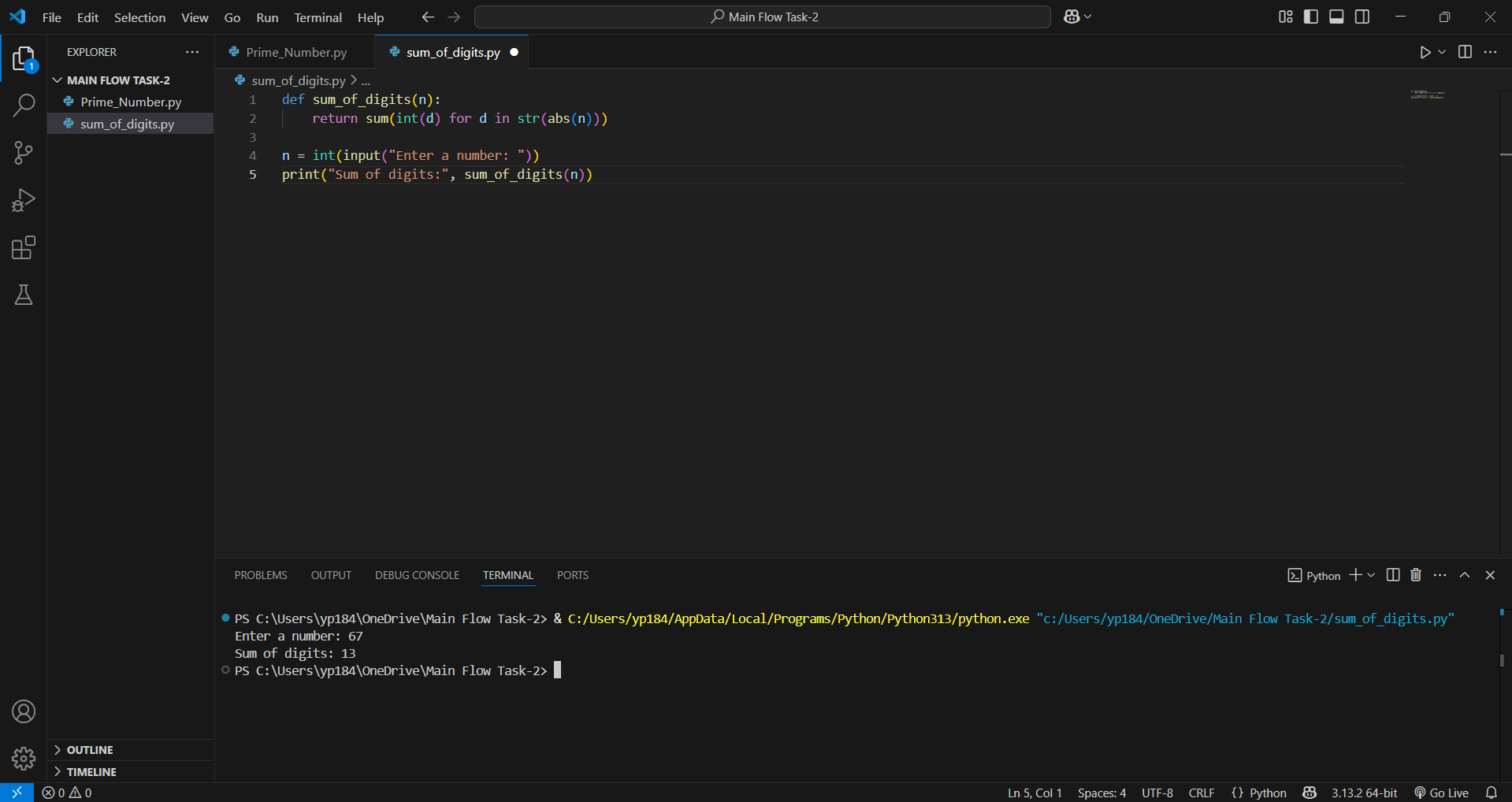
1. **Sum of Digits**: Adds all digits in a number using string conversion and iteration.

Code: def sum\_of\_digits(n):

return sum(int(d) for d in str(abs(n)))

* 1. = int(input("Enter a number: ")) print("Sum of digits:", sum\_of\_digits(n))

Screenshot:



1. **LCM and GCD:** Uses math.gcd() to compute both values efficiently.

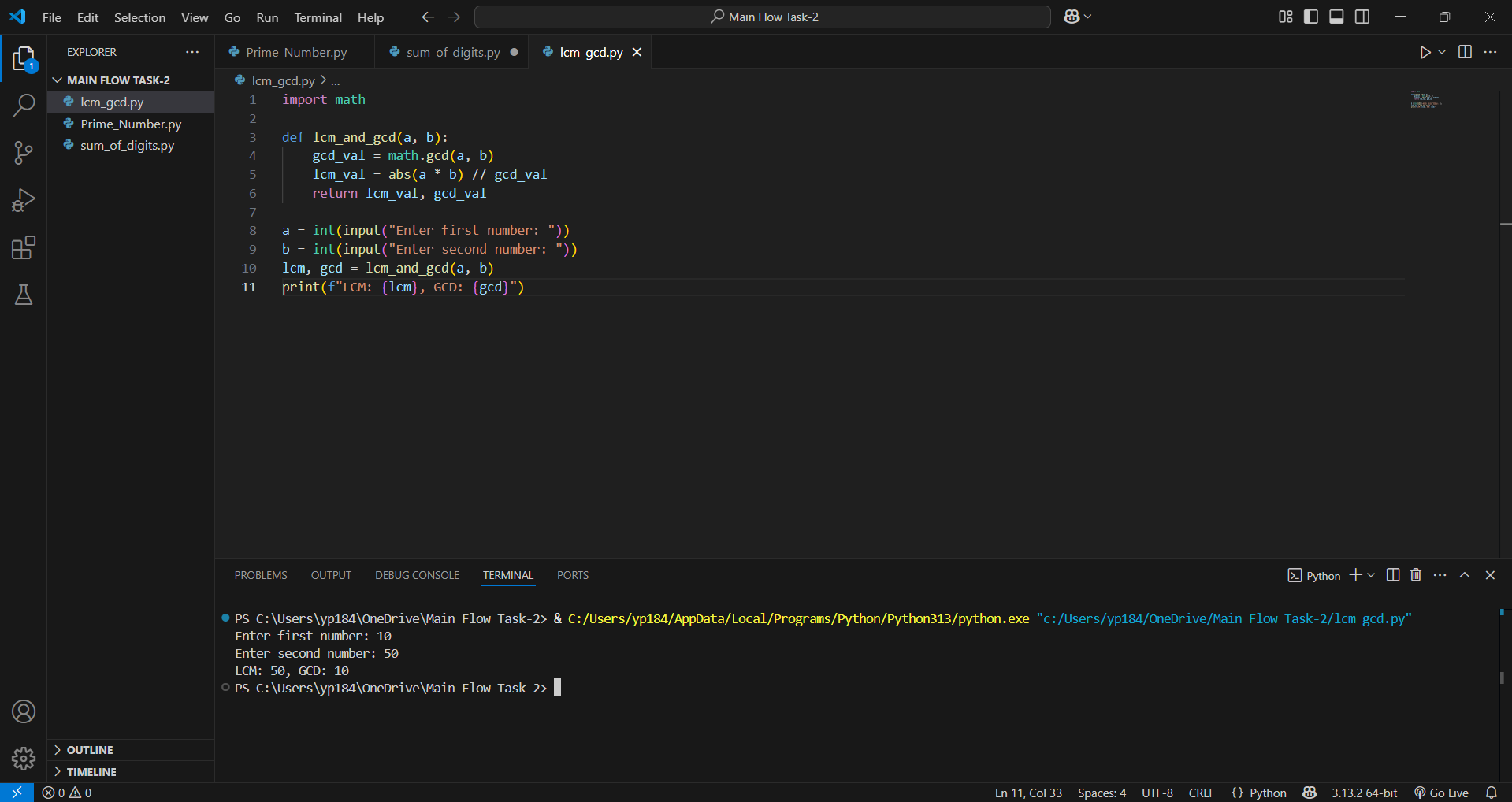
Code: import math

def lcm\_and\_gcd(a, b):

gcd\_val = math.gcd(a, b) lcm\_val = abs(a \* b) // gcd\_val return lcm\_val, gcd\_val

a = int(input("Enter first number: ")) b = int(input("Enter second number: ")) lcm, gcd = lcm\_and\_gcd(a, b) print(f"LCM: {lcm}, GCD: {gcd}")

Screenshot:



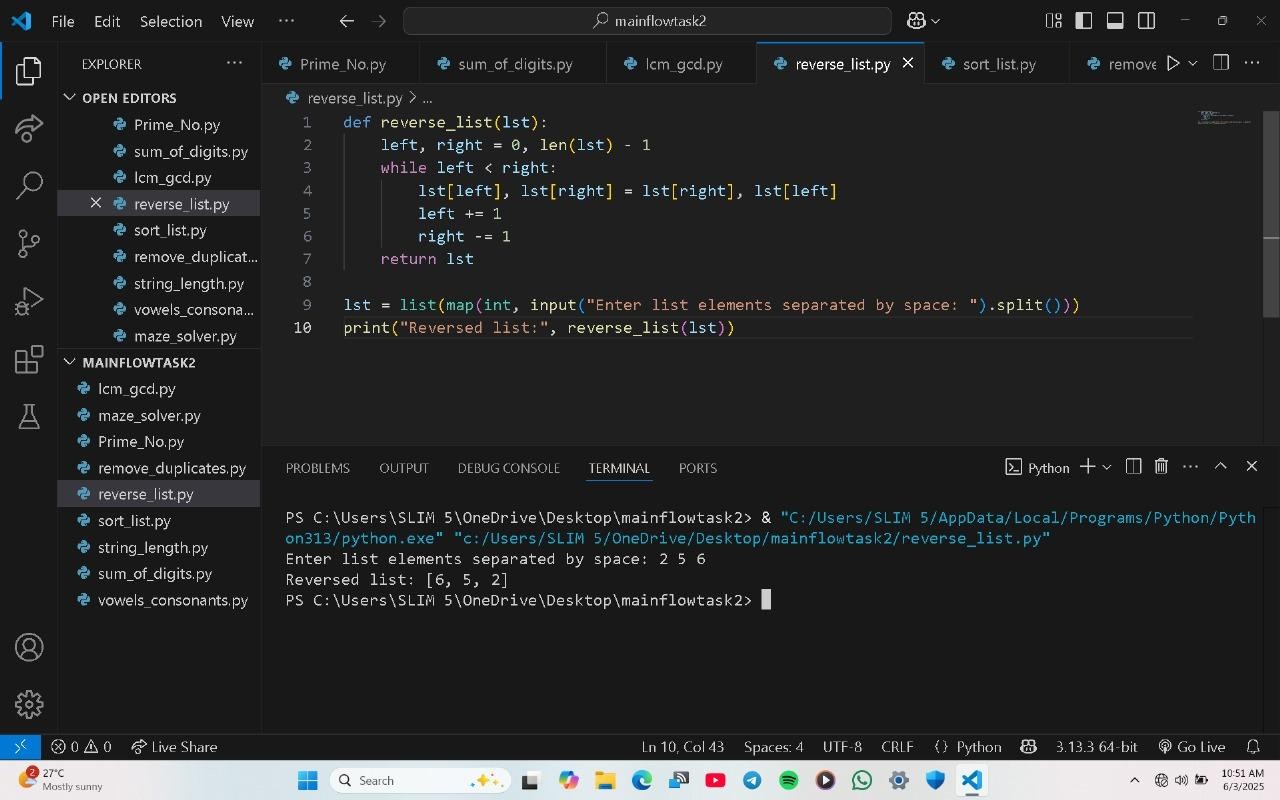
1. **List Reversal**: Reverses a list manually without using built-in functions.

Code: def reverse\_list(lst): left, right = 0, len(lst) - 1 while left < right:

lst[left], lst[right] = lst[right], lst[left] left += 1 right -= 1 return lst

lst = list(map(int, input("Enter list elements separated by space: ").split())) print("Reversed list:", reverse\_list(lst))

Screenshot:



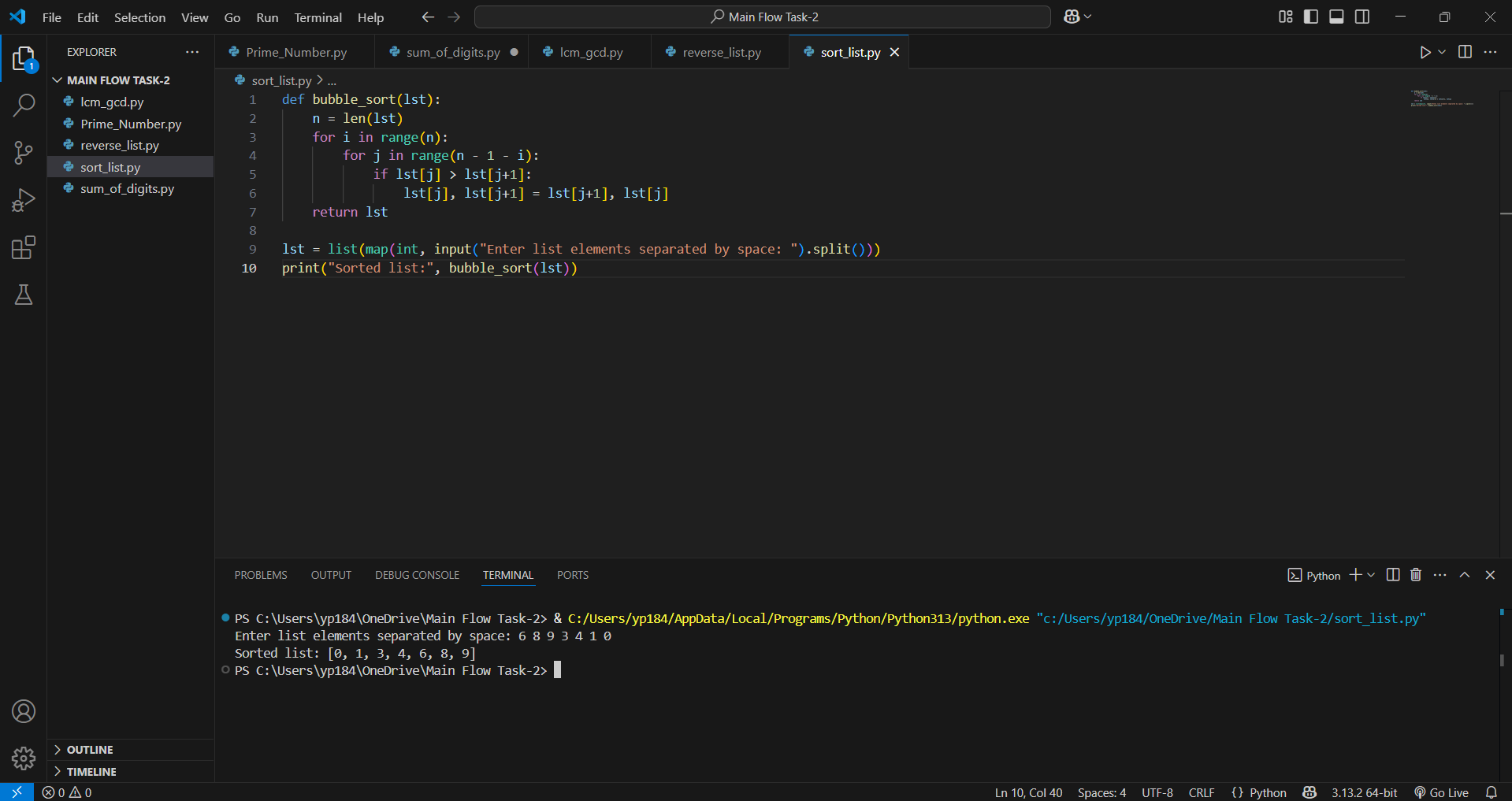
1. **Sort a List**: Implements Bubble Sort for ascending order sorting.

Code : def bubble\_sort(lst):

* 1. = len(lst) for i in range(n): for j in range(n - 1 - i): if lst[j] > lst[j + 1]:

lst[j], lst[j + 1] = lst[j + 1], lst[j] return lst

lst = list(map(int, input("Enter list elements separated by space: ").split())) print("Sorted list:", bubble\_sort(lst)) Screenshot:



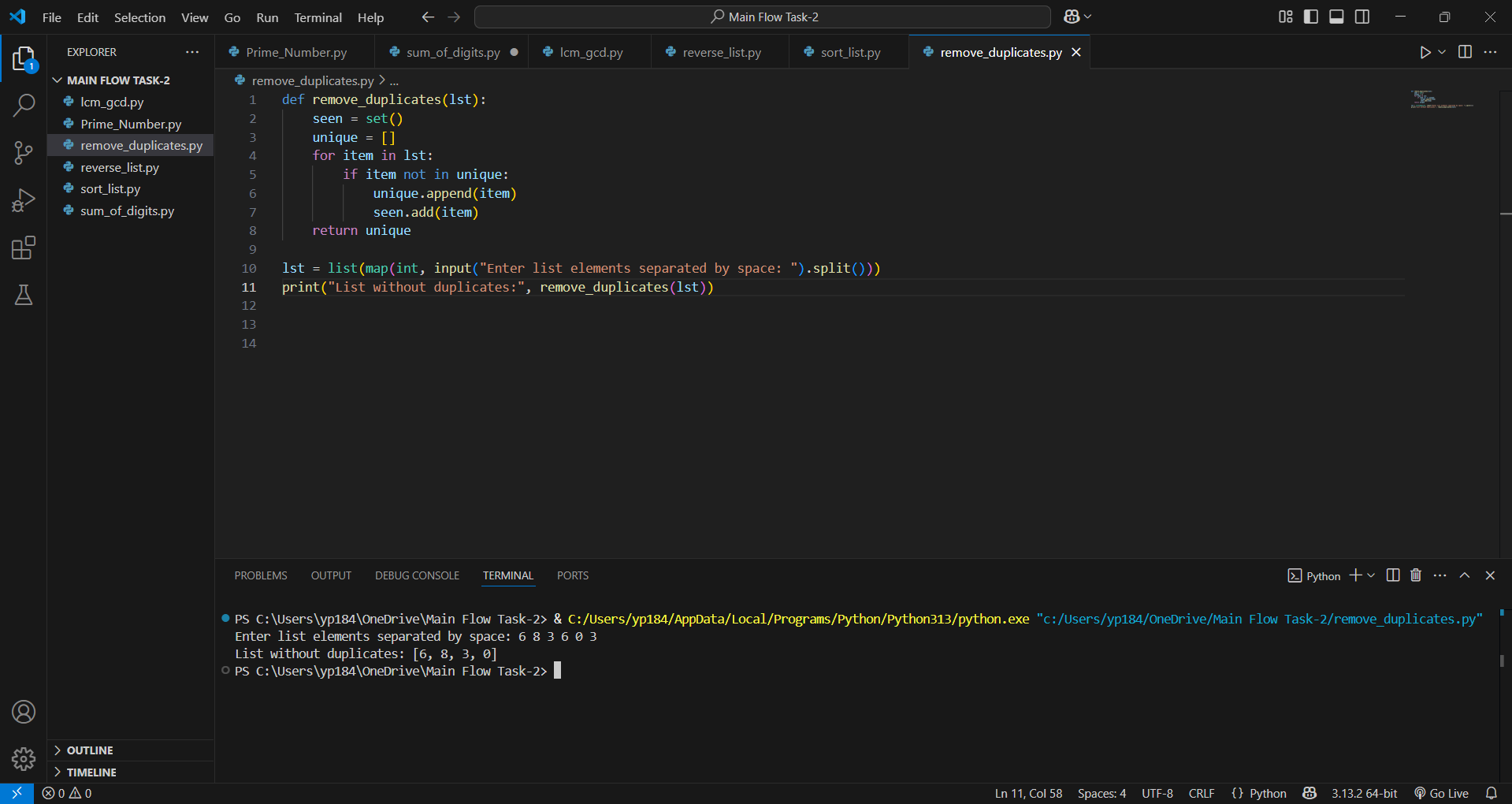
1. **Remove Duplicates**: Removes duplicate elements while preserving original order using sets.

Code: def remove\_duplicates(lst):

seen = set() unique = [] for item in lst: if item not in seen: unique.append(item) seen.add(item) return unique

lst = list(map(int, input("Enter list elements separated by space: ").split())) print("List without duplicates:", remove\_duplicates(lst))

Screenshot:



1. **String Length**: Counts characters manually without using len().

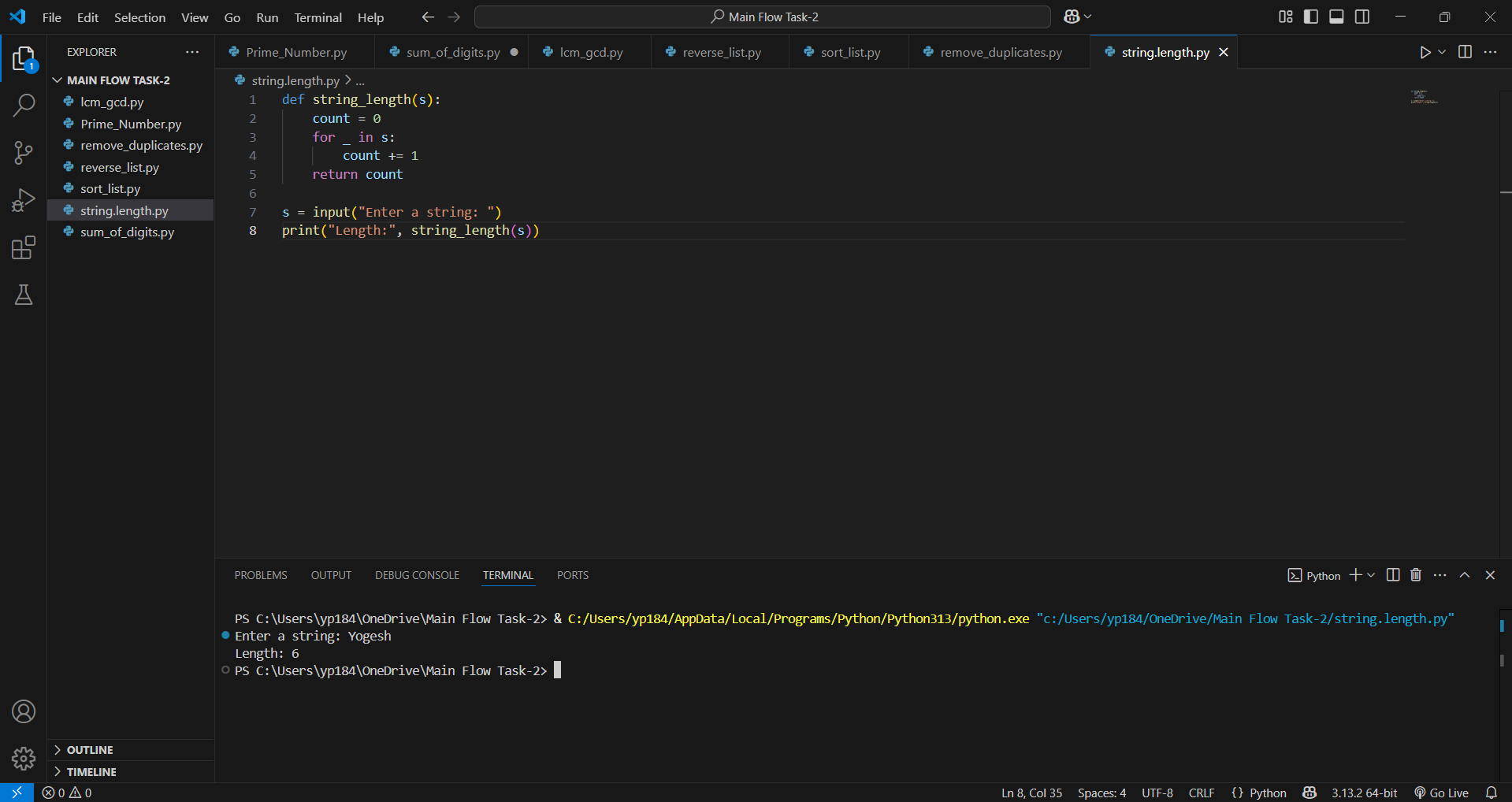
Code: def string\_length(s):

count = 0 for \_ in s:

count += 1 return count

* 1. = input("Enter a string: ") print("Length:", string\_length(s))

Screenshot:



1. **Count Vowels and Consonants:** Counts vowels and consonants using character checks.

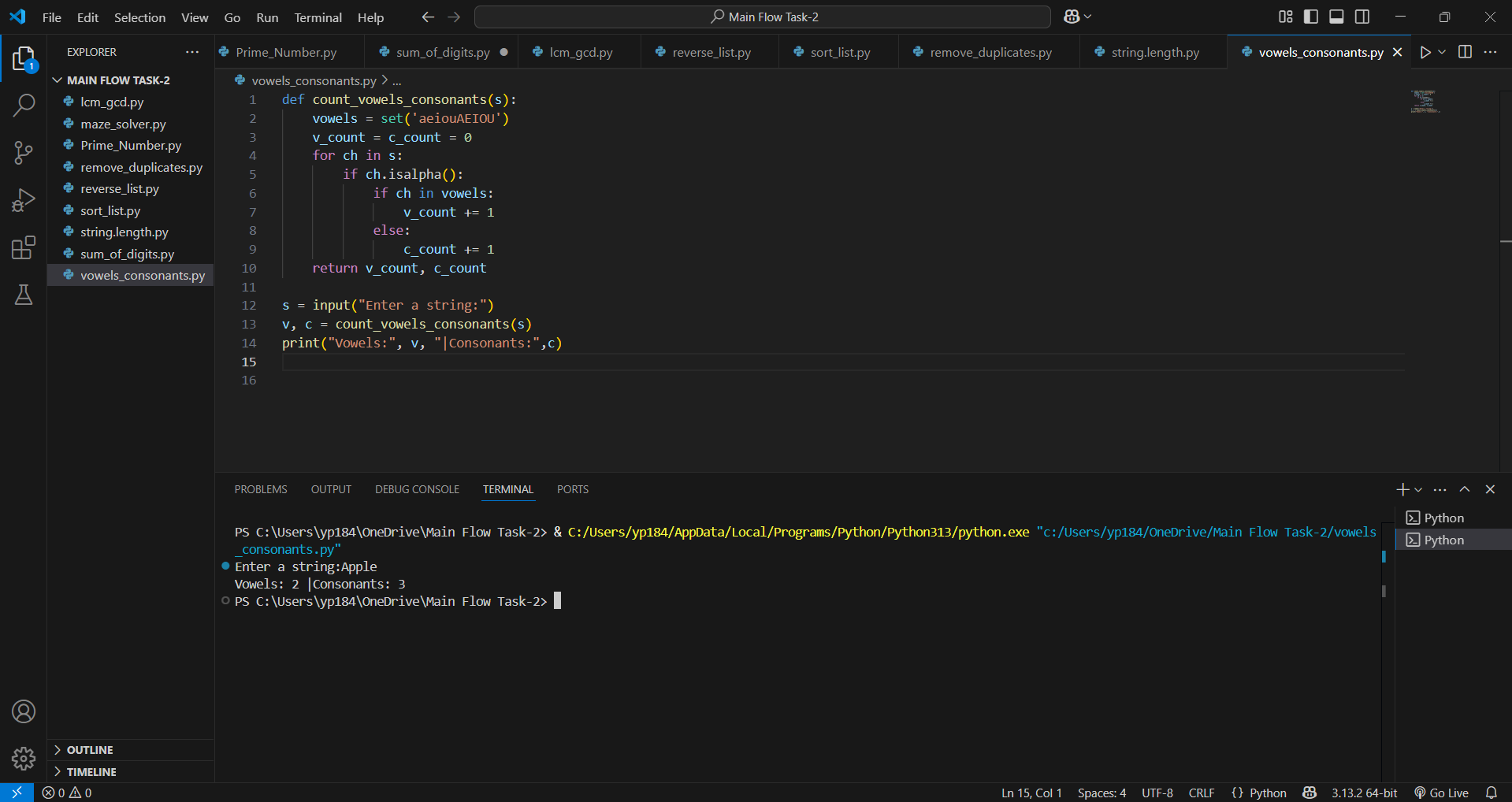
Code: def count\_vowels\_consonants(s):

vowels = set('aeiouAEIOU') v\_count = c\_count = 0 for ch in s: if ch.isalpha(): if ch in vowels: v\_count += 1 else:

c\_count += 1 return v\_count, c\_count

* 1. = input("Enter a string: ") v, c = count\_vowels\_consonants(s) print("Vowels:", v, "| Consonants:", c)

Screenshot:



**2. Maze Generator and Solver (Bonus Challenge):** Creates a solvable maze using DFS and solves it recursively.

Code: import random

def generate\_maze(n, m):

maze = [[1 for \_ in range(m)] for \_ in range(n)] def carve(x, y):

directions = [(2, 0), (-2, 0), (0, 2), (0, -2)] random.shuffle(directions) for dx, dy in directions: nx, ny = x + dx, y + dy if 1 <= nx < n-1 and 1 <= ny < m-1 and maze[nx][ny] == 1:

maze[nx][ny] = 0

maze[x + dx//2][y + dy//2] = 0

carve(nx, ny) maze[1][1] = 0 carve(1, 1) maze[n-2][m-2] = 0 return maze

def solve\_maze(maze, start, end):

n, m = len(maze), len(maze[0]) path = []

visited = [[False]\*m for \_ in range(n)] def dfs(x, y):

if not (0 <= x < n and 0 <= y < m) or maze[x][y] == 1 or visited[x][y]:

return False path.append((x, y)) visited[x][y] = True if (x, y) == end: return True if (dfs(x+1, y) or dfs(x-1, y) or dfs(x, y+1) or dfs(x, y-1)):

return True path.pop() return False dfs(\*start) return path

def print\_maze(maze, path=[]): for i in range(len(maze)):

row = "" for j in range(len(maze[0])): if (i, j) in path: row += "." else:

row += "#" if maze[i][j] == 1 else " " print(row)

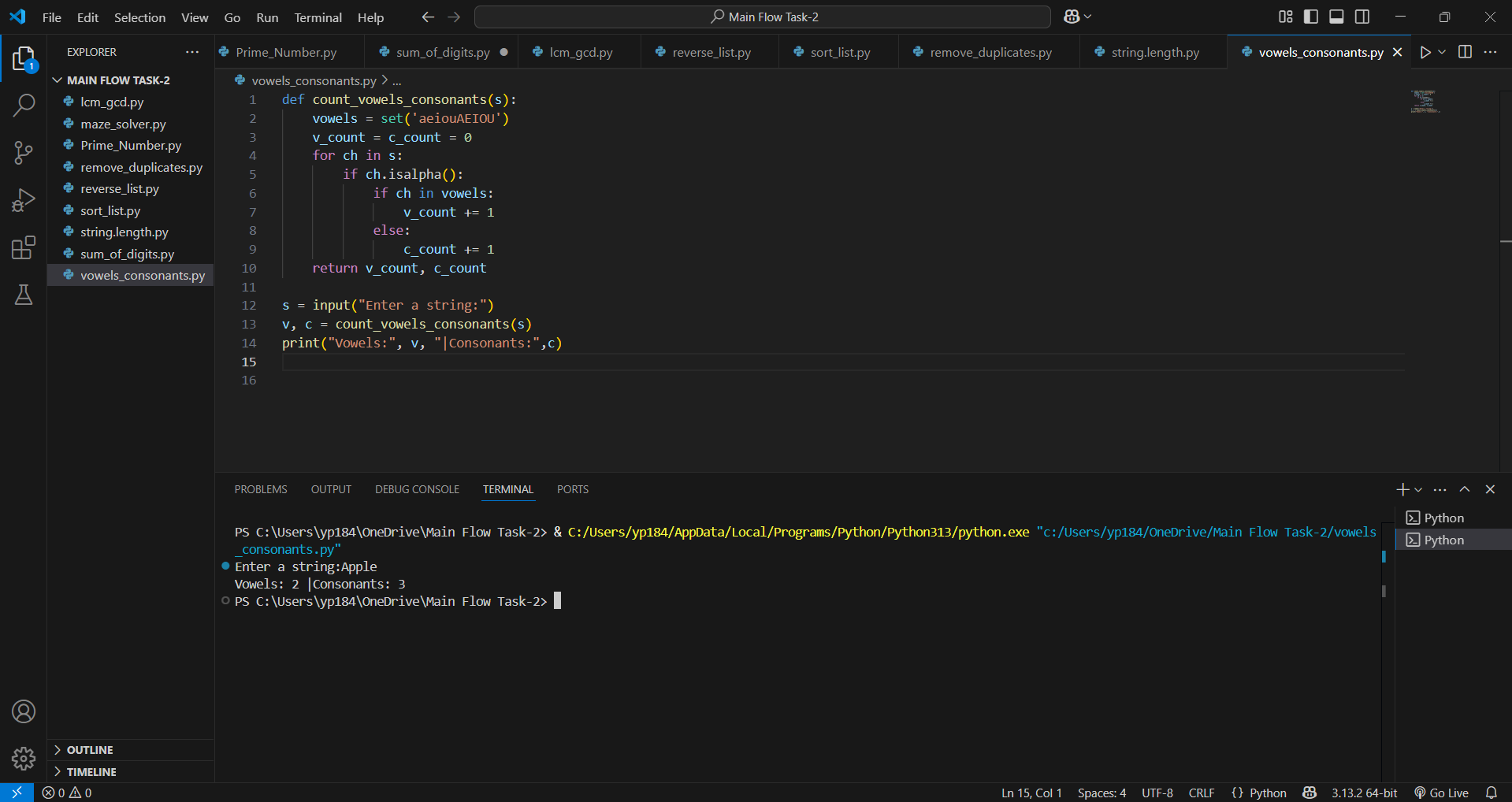
# === User Input Section === print("Maze Generator and Solver") n = int(input("Enter number of rows (odd number ≥ 5): ")) m = int(input("Enter number of columns (odd number ≥ 5): "))

maze = generate\_maze(n, m)

path = solve\_maze(maze, (1, 1), (n - 2, m - 2))

print("Generated Maze with Solution Path:") print\_maze(maze, path)

Screenshot:



# 4. Technologies and Skills Used

* Python (core language)
* Algorithms: DFS, recursion, loops, mathematical formulas
* Data Structures: Lists, Sets, Strings
* Problem-Solving and Logical Thinking
* Terminal-based Visualization Techniques

# 5. Challenges Faced

* Avoiding built-in functions made implementation more logical.
* Maze generation and solving was a complex yet rewarding challenge.
* Time management to complete within the 7-day deadline.

# 6. Learning Outcomes

* Improved ability to write clean, efficient, and logical Python code.
* Deeper understanding of common problem-solving methods.
* Boost in confidence to tackle advanced Python projects.
* Exposure to real software development constraints like deadlines and optimization.

# 7. Future Enhancements

* Extend the maze program with BFS.
* Add a user menu interface to test all features.
* Create GUI versions using Tkinter or PyGame.

# 8. Conclusion

Completing Python Task-2 under the guidance of Main Flow Services and Technologies Pvt. Ltd. has been a rewarding experience. This task has strengthened my Python fundamentals and enhanced my confidence to solve more complex problems.

Thank you to Main Flow Services and Technologies for the opportunity and mentorship!