SR UNIVERSITY

AI ASSIST CODING

Lab- 1: Environment Setup – GitHub Copilot and VS Code Integration

**Name**: k srikar

**Roll No:**2503A51L03

**BATCH *NO:***19

Lab Objectives:

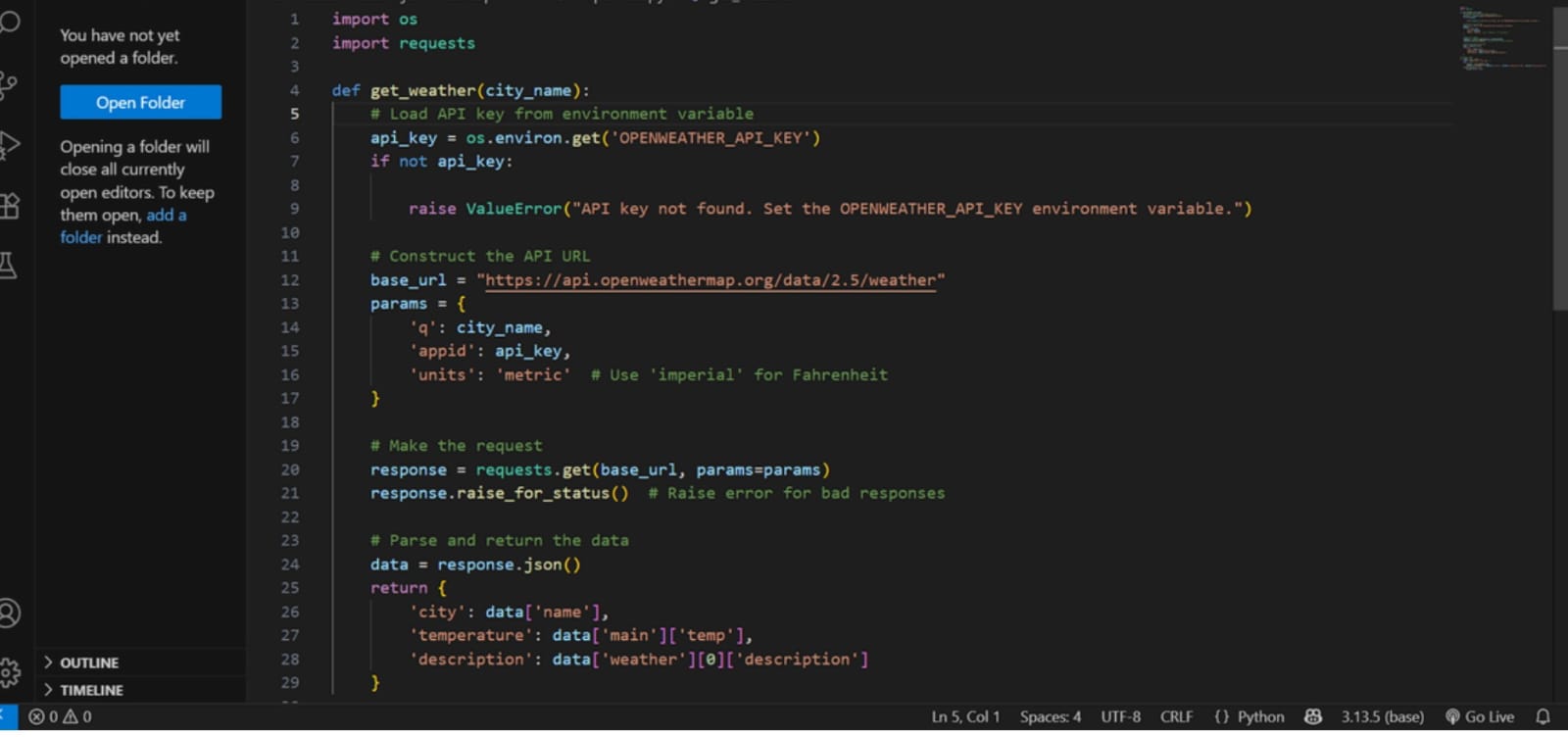
* To install and configure GitHub Copilot in Visual Studio Code
* To explore AI-assisted code generation using GitHub Copilot.
* To analyze the accuracy and effectiveness of Copilot's code suggestions.
* To understand prompt-based programming using comments and code context

Lab Outcomes (LOs):

After completing this lab, students will be able to:

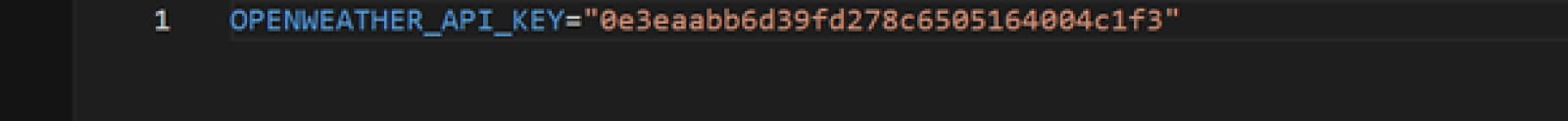
* Set up GitHub Copilot in VS Code successfully.
* Use inline comments and context to generate code with Copilot.
* Evaluate AI-generated code for correctness and readability.
* Compare code suggestions based on different prompts and programming styles.

Environment variable setup code:

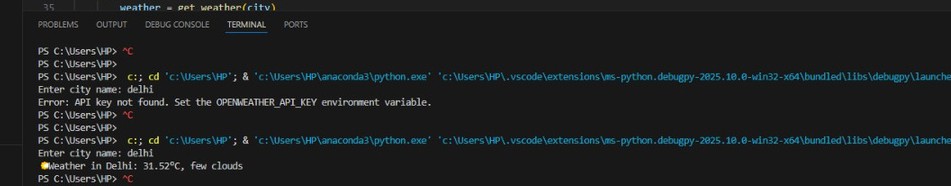




Generated Weather API Key:



After Environment Setup:

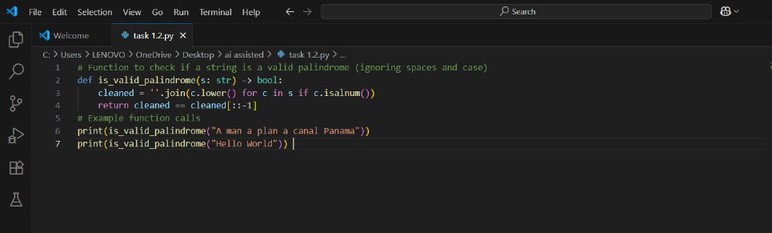


Task #1:

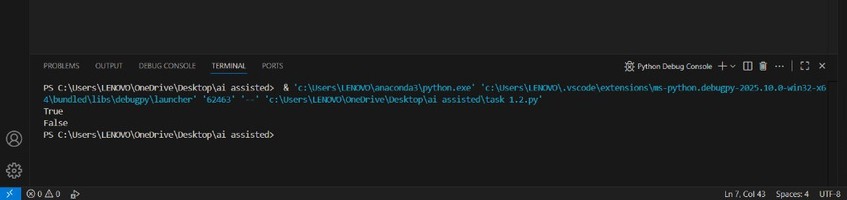
Prompt:

* Write a comment: # Function to check if a string is a valid palindrome (ignoring spaces and case) and allow Copilot to complete it.

Code Generated:



Output After executing Code:



Your Observations:

The observation from the above code is:

 The function is\_valid\_palindrome checks if a given string is a palindrome, ignoring spaces, punctuation, and case.

 When called with "A man a plan a canal Panama", it returns True because this phrase is a palindrome when spaces and case are ignored.

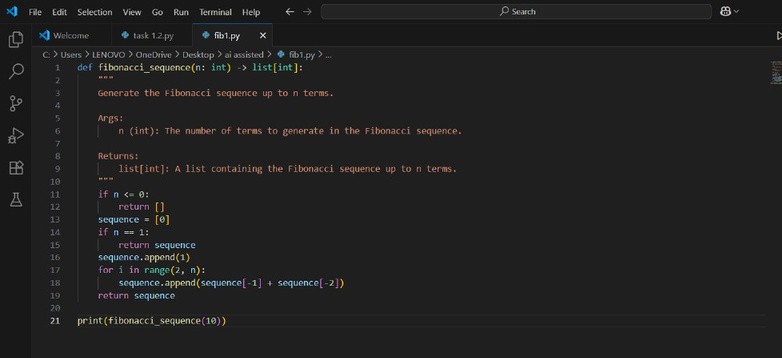
 When called with "Hello World", it returns False because this phrase is not a palindrome.

# TASK #2:

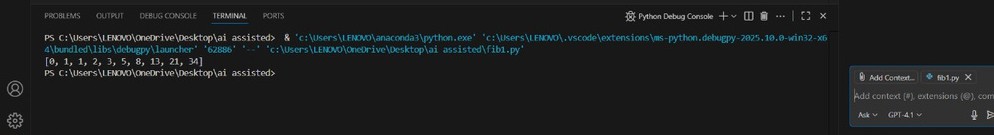
## Prompt:

* Generate a Python function that returns the Fibonacci sequence up to n terms. Prompt with only a function header and docstring

Code Generated:



Output After executing Code:



Your Observations:

1. Correct Fibonacci Logic:

 The function correctly generates the Fibonacci sequence using iteration. It handles edge cases like n <= 0 and n == 1.

1. Python Type Hints Used:

 The function signature uses type hints (n: int -> list[int]), which improves code readability and helps with static analysis tools.

1. Docstring Included:

 There’s a detailed docstring explaining the arguments and return type, which is great for documentation and usability.

1. Edge Case Handling:

 The function checks for non-positive n and handles n == 1 separately, preventing index errors.

1. Clean and Readable Code:

 Indentation, spacing, and variable naming are clear and follow Python conventions.

1. Execution Output Verifled:

 The terminal shows the correct output of the first 10 Fibonacci numbers

# TASK #3:

## Prompt:

* Write a comment like # Function to reverse a string and use Copilot to generate the function.

Code Generated:



Output After executing Code:



Observations:

1. Correct String Reversal Logic:

 The use of Python slicing [::-1] is a concise and efficient way to reverse a string.

1. Simple and Clean Implementation:

 The code is minimal, readable, and directly focuses on the core task of reversing a string.

1. Appropriate Function Use:

 The logic is wrapped inside a function (reverse\_string), which makes the code reusable.

1. Function Successfully Tested:

 The output in the terminal (dlrow olleh) confirms that the function works correctly for the input "hello world".

1. Comment for Clarity:

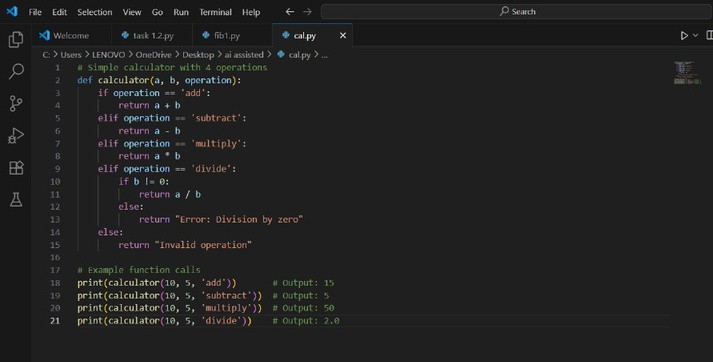
 There is a comment at the top of the script that explains the purpose of the function.

# TASK #4:

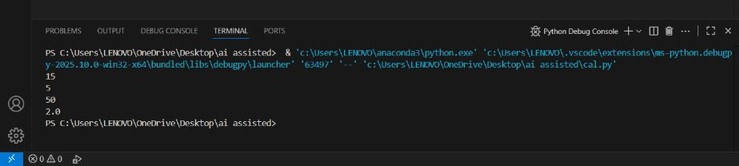
## Prompt:

 Generate a program that simulates a basic calculator (add, subtract, multiply, divide). Write the comment: # Simple calculator with 4 operations and let AI complete

Code Generated:



Output After executing Code:



Your Observations:

1. Functional Calculator Implementation:

The calculator function handles the four basic arithmetic operations: addition, subtraction, multiplication, and division.

1. Input Flexibility:

 The function takes three parameters: two numbers and a string indicating the operation — simple and user- friendly design.

1. Division by Zero Check:

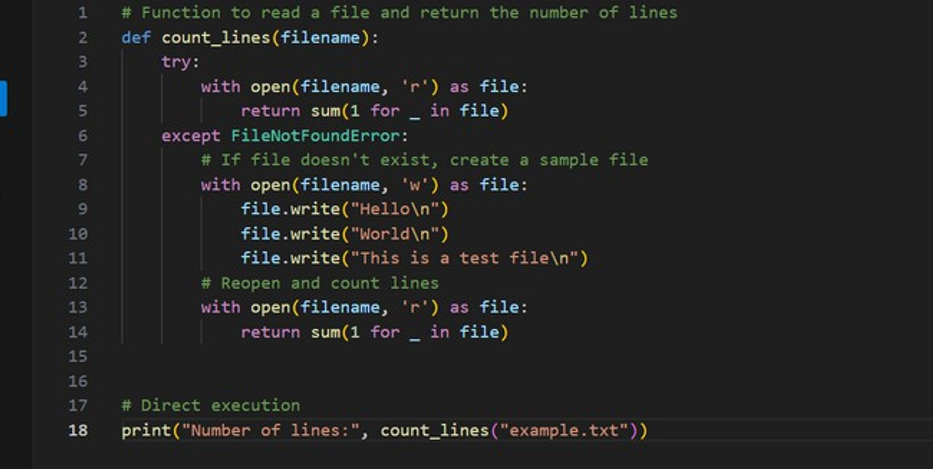
 Great job handling the divide case carefully by checking if b != 0 to avoid runtime errors.

# TASK #5:

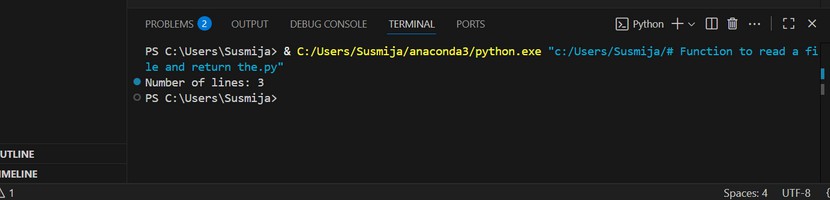
## Prompt:

 Use a comment to instruct AI to write a function that reads a file and returns the number of lines

Code Generated:



Output After executing Code:



Your Observations:

1. Correct Function Purpose:

 The function count\_lines(filename) is designed to:  Read a file and count its number of lines.

 If the file doesn’t exist, it creates a default one and then counts the lines.

### Proper Use of Exception Handling:

 The try-except block catches a File Not Found Error and handles it gracefully by creating a sample file.

This prevents the program from crashing due to a missing file.

1. Efficient Line Counting Logic:

 Uses generator expression to count lines:

 sum(1 for \_ in file)

 This is memory-efficient and Pythonic.

### Sample File Created with 3 Lines:

 The file is created with 3 specific lines:  Hello

 World

 This is a test file

### Clear Output Statement:

 Displays the result to the user using:

 print("Number of lines:", count\_lines("example.txt")