AI ASSISTED CODING

Roll no.: 2503A51L41

Batch: 24BTCAICSB20

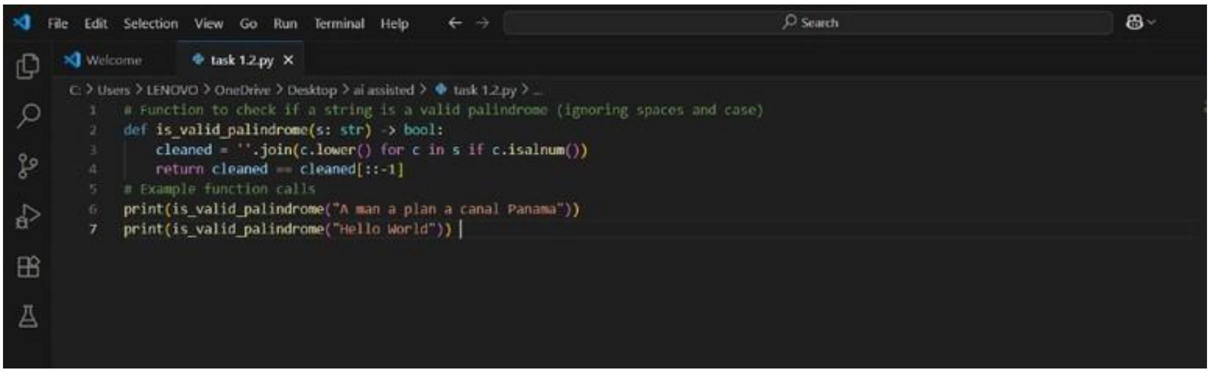
Name: G Hasini

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

Task #1

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

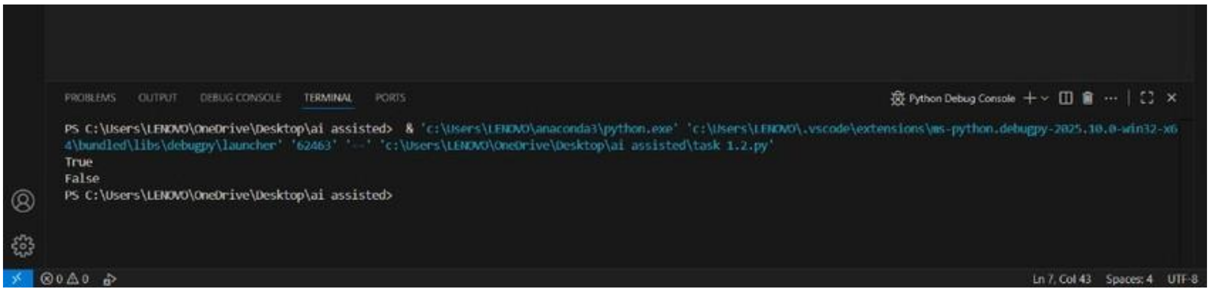
Code generated:



File Edit Selection View Go Run Terminal Help
Search
Welcome
task 1.2.py
C: > Users > LENOVO > OneDrive > Desktop > ai assisted > task 1.2.py > ...

```python
# Function to check if a string is a valid palindrome (ignoring spaces and case)
def is_valid_palindrome(s: str) -> bool:
    cleaned = ''.join(c.lower() for c in s if c.isalnum())
    return cleaned == cleaned[::-1]
# Example function calls
print(is_valid_palindrome("A man a plan a canal Panama"))
print(is_valid_palindrome("Hello World"))
```

Output:



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Debug Console
Ln 7, Col 43 Spaces: 4 UTF-8

```
PS C:\Users\LENOVO\OneDrive\Desktop\ai assisted>  & 'c:\Users\LENOVO\anaconda3\python.exe' 'c:\Users\LENOVO\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x6
4\bundled\libs\debugpy\launcher' '62463' '--' 'c:\Users\LENOVO\OneDrive\Desktop\ai assisted\task 1.2.py'
True
False
PS C:\Users\LENOVO\OneDrive\Desktop\ai assisted>
```

Observation:

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

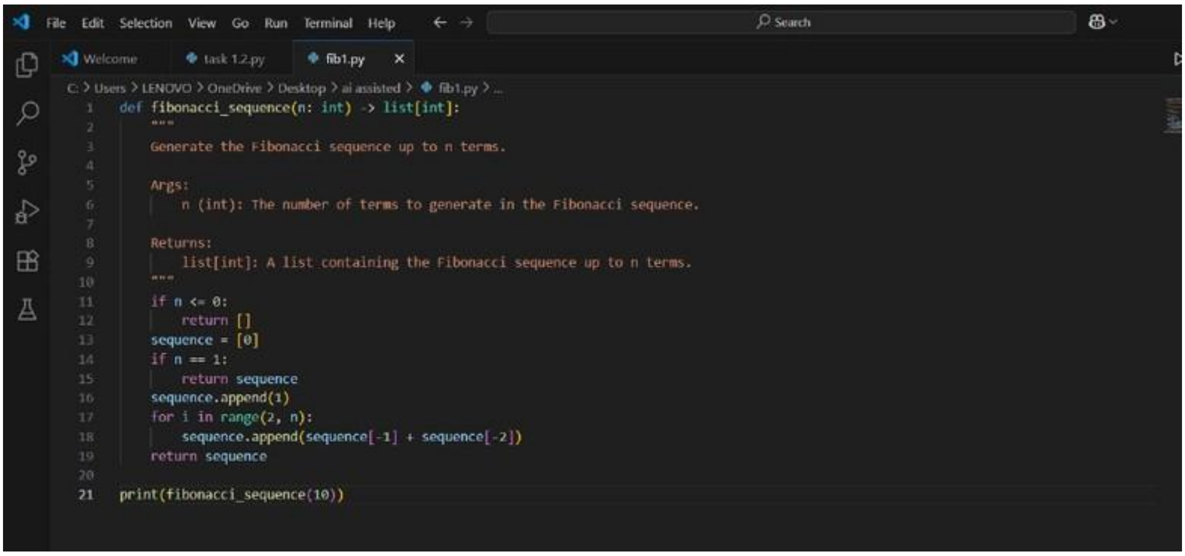
- When called with "A man a plan a canal Panama", it returns True because this phrase is a palindrome when spaces and case are ignored.

- 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:



File
Edit
Selection
View
Go
Run
Terminal
Help
Search
Welcome
task 1.2.py
fib1.py
C: > Users > LENOVO > OneDrive > Desktop > ai assisted > fib1.py > ...

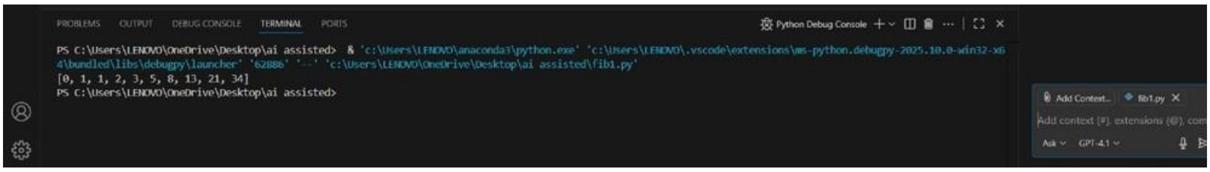
```python
def fibonacci_sequence(n: int) -> list[int]:
    """
    Generate the Fibonacci sequence up to n terms.

    Args:
        n (int): The number of terms to generate in the Fibonacci sequence.

    Returns:
        list[int]: A list containing the Fibonacci sequence up to n terms.
    """
    if n <= 0:
        return []
    sequence = [0]
    if n == 1:
        return sequence
    sequence.append(1)
    for i in range(2, n):
        sequence.append(sequence[-1] + sequence[-2])
    return sequence

print(fibonacci_sequence(10))
```

Output:



```
PS C:\Users\LENOVO\OneDrive\Desktop\ai assisted>  & 'c:\Users\LENOVO\anaconda3\python.exe' 'c:\Users\LENOVO\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x6
4\bundled\libs\debugpy\launcher' '62886' '--' 'c:\Users\LENOVO\OneDrive\Desktop\ai assisted\fib1.py'
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\LENOVO\OneDrive\Desktop\ai assisted>
```

Observation:

1. Correct Fibonacci Logic:

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

2. Python Type Hints Used:

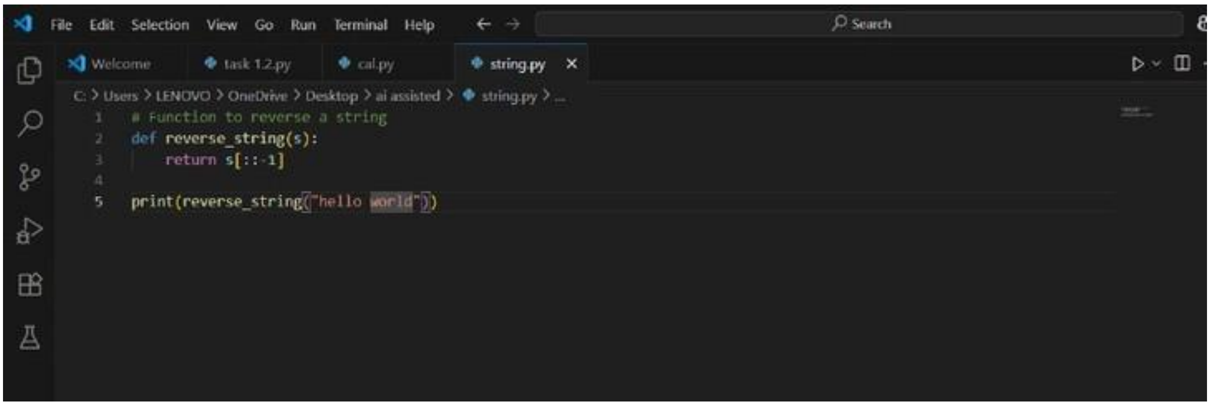
   The function signature uses type hints (n: int -> list[int]), which improves code readability and helps with static analysis tools.

3. Docstring Included:
   - There's a detailed docstring explaining the arguments and return type, which is great for documentation and usability.
4. Edge Case Handling:
   - The function checks for non-positive n and handles n == 1 separately, preventing index errors.
5. Clean and Readable Code:
   - Indentation, spacing, and variable naming are clear and follow Python conventions.
6. Execution Output Verified:
   - 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.

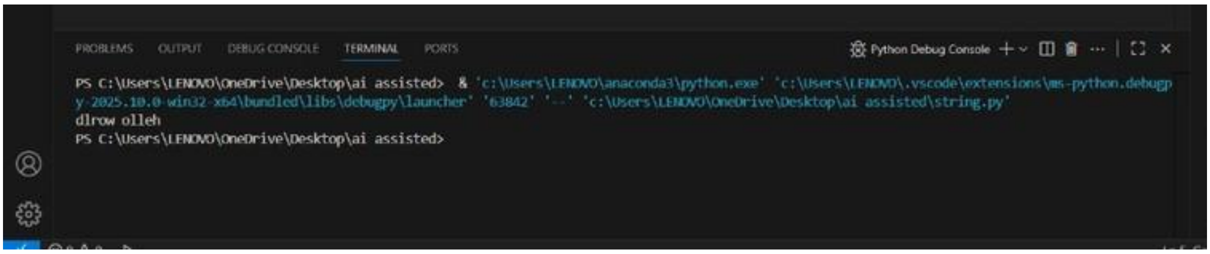
Code generated:


File Edit Selection View Go Run Terminal Help
Search
Welcome
task 1.2.py
cal.py
string.py
C: > Users > LENOVO > OneDrive > Desktop > ai assisted > string.py > ...

```python
# Function to reverse a string
def reverse_string(s):
    return s[::-1]

print(reverse_string("hello world"))
```

Output:



```
PS C:\Users\LENOVO\OneDrive\Desktop\ai assisted>  & 'c:\Users\LENOVO\anaconda3\python.exe' 'c:\Users\LENOVO\.vscode\extensions\ms-python.debugp
y-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '63842' '--' 'c:\Users\LENOVO\OneDrive\Desktop\ai assisted\string.py'
dlrow olleh
PS C:\Users\LENOVO\OneDrive\Desktop\ai assisted>
```

Observation:

1. Correct String Reversal Logic:

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

2. Simple and Clean Implementation:

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

3. Appropriate Function Use:

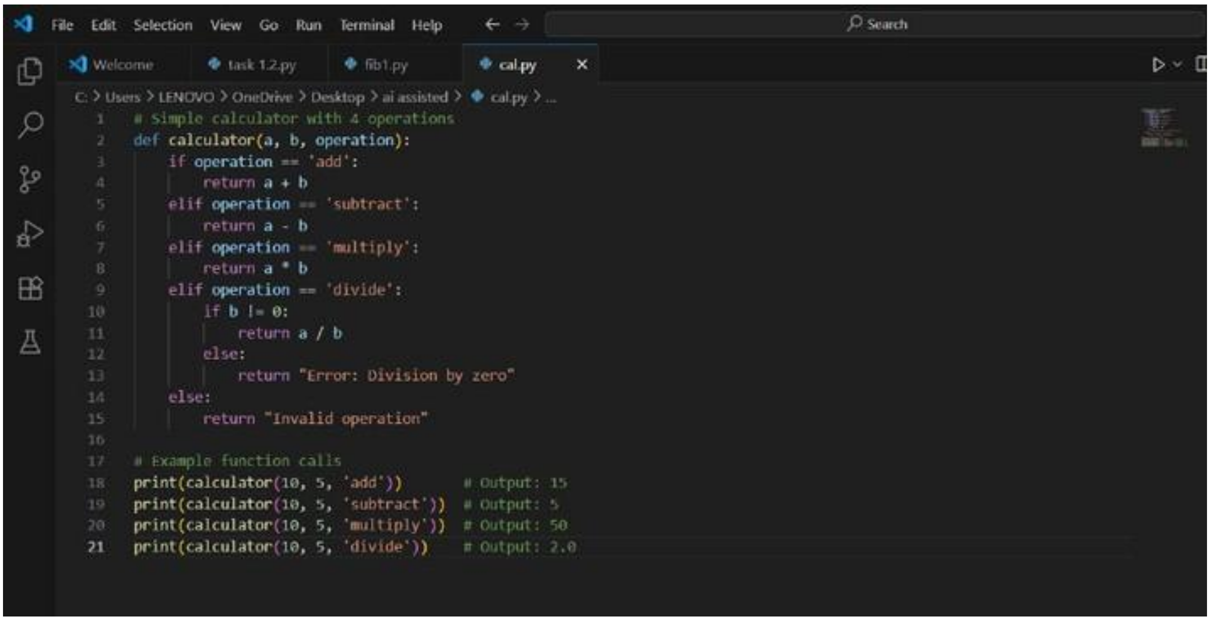
○ The logic is wrapped inside a function (reverse_string), which makes the code reusable.

4. Function Successfully Tested:

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 it.

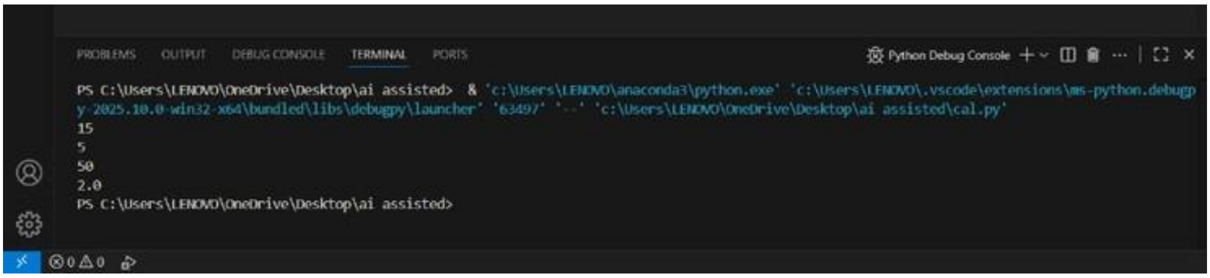
Code generated:



```python
# Simple calculator with 4 operations
def calculator(a, b, operation):
    if operation == 'add':
        return a + b
    elif operation == 'subtract':
        return a - b
    elif operation == 'multiply':
        return a * b
    elif operation == 'divide':
        if b != 0:
            return a / b
        else:
            return "Error: Division by zero"
    else:
        return "Invalid operation"

# Example function calls
print(calculator(10, 5, 'add'))       # Output: 15
print(calculator(10, 5, 'subtract'))  # Output: 5
print(calculator(10, 5, 'multiply'))  # Output: 50
print(calculator(10, 5, 'divide'))    # Output: 2.0
```

Output:



```
PS C:\Users\LENOVO\OneDrive\Desktop\ai assisted>  & 'c:\Users\LENOVO\anaconda3\python.exe' 'c:\Users\LENOVO\.vscode\extensions\ms-python.debugp
y-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '63497' '--' 'c:\Users\LENOVO\OneDrive\Desktop\ai assisted\cal.py'
15
5
50
2.0
PS C:\Users\LENOVO\OneDrive\Desktop\ai assisted>
```

Observation:

Functional Calculator Implementation:

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

Input Flexibility:

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

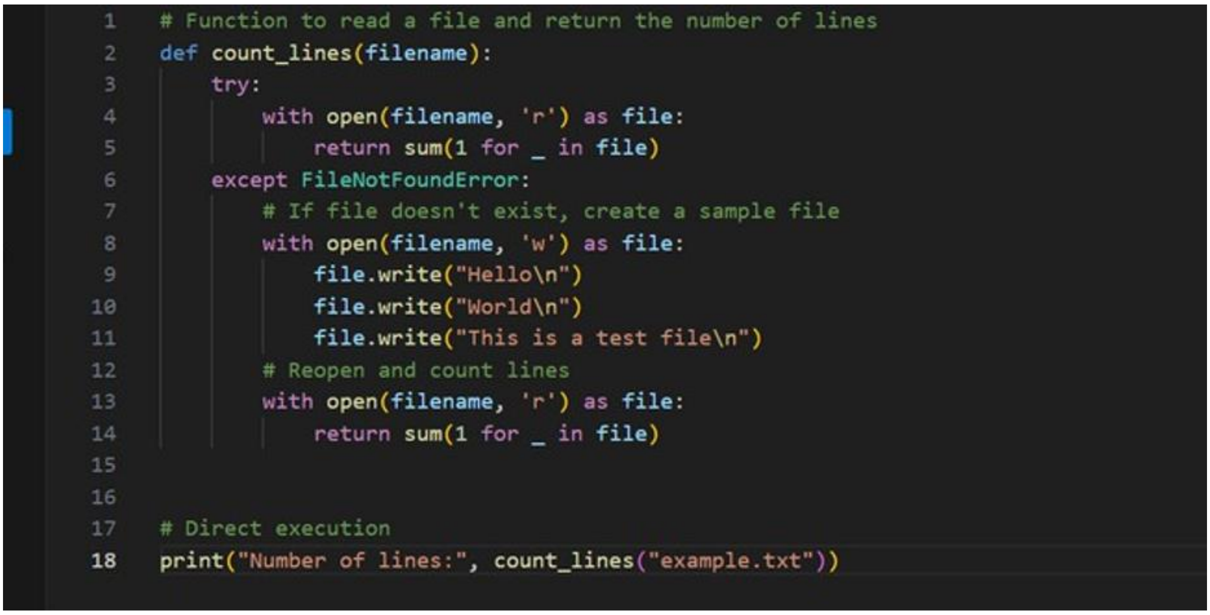
Division by Zero Check:

- 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.

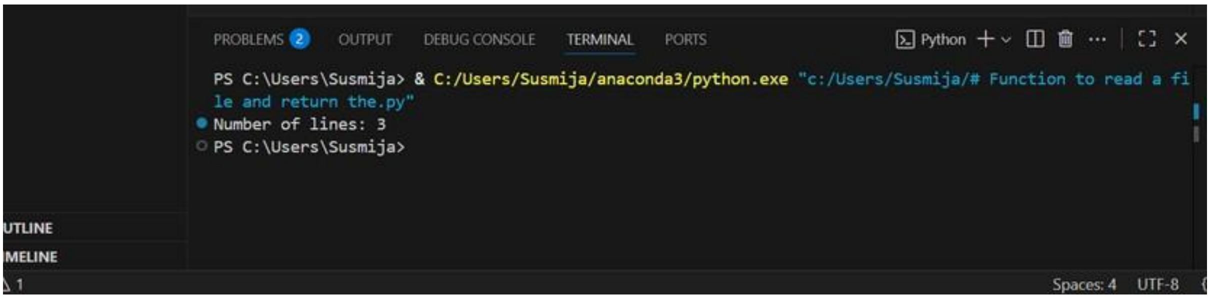
Code generated:



```python
# Function to read a file and return the number of lines
def count_lines(filename):
    try:
        with open(filename, 'r') as file:
            return sum(1 for _ in file)
    except FileNotFoundError:
        # If file doesn't exist, create a sample file
        with open(filename, 'w') as file:
            file.write("Hello\n")
            file.write("World\n")
            file.write("This is a test file\n")
        # Reopen and count lines
        with open(filename, 'r') as file:
            return sum(1 for _ in file)


# Direct execution
print("Number of lines:", count_lines("example.txt"))
```

Output:



```
PS C:\Users\Susmija> & C:/Users/Susmija/anaconda3/python.exe "c:/Users/Susmija/# Function to read a fi
le and return the.py"
Number of lines: 3
PS C:\Users\Susmija>
```

Observation:

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.

2.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.

3.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")