AI-ASSISTED CODING LAB

ASSIGNMENT 1
ENROLLMENT NO: 2503A51I44
BATCH NO: 20
NAME: MEER BURHAN ALI HASHMI

TASK 1

Description:

Write a comment:
Function to check if a string is a valid palindrome (ignoring spaces and case)
and allow Copilot to generate the function implementation.

Prompt:

Generate a comment:
Function to check if a string is a valid palindrome (ignoring spaces and case)
and let the AI complete it.

Code:

```python
# Function to check if a string is a valid palindrome (ignoring spaces and case)
def is_valid_palindrome(s):
    # Remove spaces and convert to lowercase
    cleaned = ''.join(c.lower() for c in s if c.isalnum())
    # Check if the cleaned string is equal to its reverse
    return cleaned == cleaned[::-1]

# Example usage
print(is_valid_palindrome("A man a plan a canal Panama"))  # Should return True
```

Output:

```
PS C:\Users\khaja\Downloads\a1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/a1/t1.py
True
PS C:\Users\khaja\Downloads\a1>
```

TASK 2

Description:

Create a Python function to return the Fibonacci sequence up to n terms. Begin with only a function header and a descriptive docstring.

Prompt:

Generate a Python function that returns the Fibonacci sequence up to n terms. The function should only include the header and a detailed docstring initially, and then complete it with an example implementation.

```python
def fibonacci_sequence(n):
    """
    Returns the Fibonacci sequence up to n terms.

    The Fibonacci sequence is a series of numbers where each number is the sum of the two pr
    starting from 0 and 1. This function generates a list containing the first n terms of th

    Parameters:
        n (int): The number of terms to generate. Must be a non-negative integer.

    Returns:
        list: A list containing the Fibonacci sequence up to n terms.

    Example:
        >>> fibonacci_sequence(5)
        [0, 1, 1, 2, 3]

        >>> fibonacci_sequence(1)
        [0]
```

Code:

Output:

```
PS C:\Users\khaja\Downloads\a1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/a1/t2.py
[0, 1, 1, 2, 3, 5, 8]
PS C:\Users\khaja\Downloads\a1>
```

TASK 3

Description:

Write a comment such as
Function to reverse a string
and use Copilot to auto-generate the corresponding function.

Prompt:

Write a comment like
Function to reverse a string
and let the AI generate the full function.

Code:

```python
# Function to reverse a string
def reverse_string(s):
    """Returns the reverse of the input string s."""
    return s[::-1]

# Example usage
print(reverse_string("hello"))  # Output: "olleh"
```

Output:

```python
# Function to reverse a string
def reverse_string(s):
    """Returns the reverse of the input string s."""
    return s[::-1]

# Example usage
print(reverse_string("hello"))  # Output: "olleh"

```

TASK 4

Description:

Create a program that simulates a simple calculator capable of performing four operations: addition, subtraction, multiplication, and division. Begin with the comment:
Simple calculator with 4 operations
and allow AI to complete it.

Prompt:

Write the comment:
Simple calculator with 4 operations
and let Copilot complete a working calculator with input/output handling and operator selection logic.

Code:

```python
# Simple calculator with 4 operations
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Error: Division by zero"
    return a / b

if __name__ == "__main__":
    print("Simple Calculator")
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")

    choice = input("Enter choice (1/2/3/4): ")

    if choice in ('1', '2', '3', '4'):
        try:
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
        except ValueError:
            print("Invalid input. Please enter numbers.")
            exit()

        if choice == '1':
            print(f"{num1} + {num2} = {add(num1, num2)}")
        elif choice == '2':
            print(f"{num1} - {num2} = {subtract(num1, num2)}")
        elif choice == '3':
            print(f"{num1} * {num2} = {multiply(num1, num2)}")
        elif choice == '4':
            print(f"{num1} / {num2} = {divide(num1, num2)}")
    else:
        print("Invalid choice.")
```

Output:

```
PS C:\Users\khaja\Downloads\a1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/a1/t5.py
Simple Calculator
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1/2/3/4): 1
Enter first number: 2
Enter second number: 3
2.0 + 3.0 = 5.0
PS C:\Users\khaja\Downloads\a1>
```

TASK 5

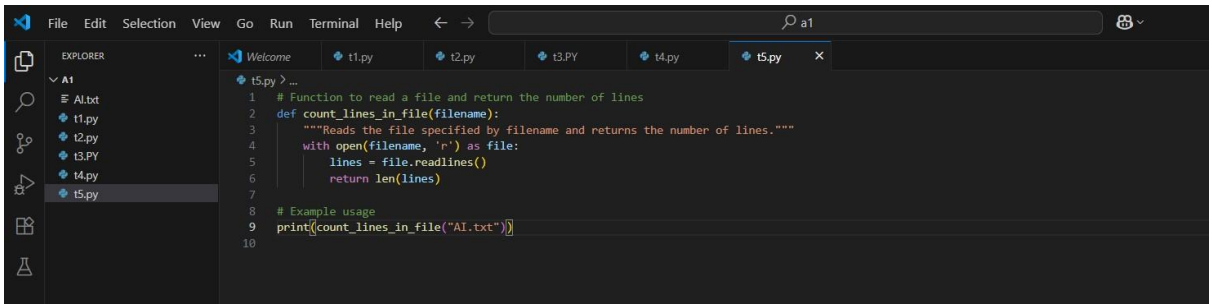
Description:

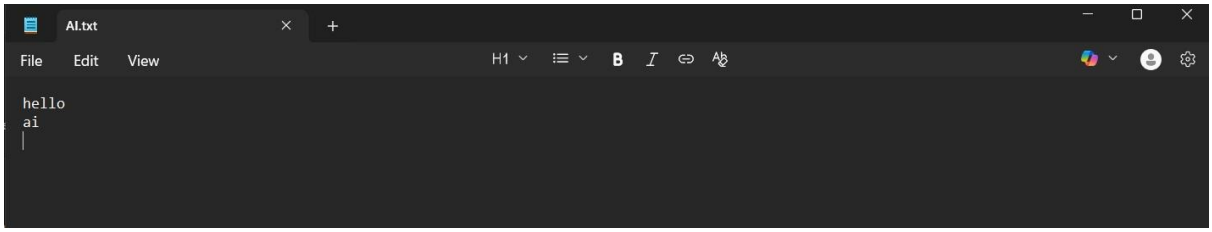
Use a comment to instruct the AI to create a function that reads a file and returns the total number of lines in it.

Prompt:

Write a comment instructing AI to write a function that reads a file and returns the number of lines.

Code:





Output:

```
PS C:\Users\khaja\Downloads\a1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/a1/t5.py
2
PS C:\Users\khaja\Downloads\a1>
```

OBSERVATION

In this lab session, I explored how GitHub Copilot can automatically generate Python code based on minimal instructions, such as comments, headers, or docstrings. This experience showed how Copilot can be a powerful tool for simplifying coding tasks and improving productivity.

- **Task 1** showed that even a brief comment about palindrome checking can guide Copilot to generate a fully functional implementation, including logic to ignore spaces and letter casing.
- **Task 2** illustrated that Copilot understands well-documented function headers and can generate accurate Fibonacci logic from just a descriptive docstring.
- **Task 3** demonstrated how a short comment about reversing a string is enough for Copilot to generate a correct and efficient function.
- **Task 4** revealed that describing a basic calculator through a comment enabled Copilot to produce an interactive, user-input-based calculator covering all four main operations.

- **Task 5** proved Copilot’s ability to handle file operations, generating a function that reads and counts lines with minimal prompting.