ASSIGNMENT-13.3

Name: V.Pranavith

HT. No: 2303A51488

Batch : 08

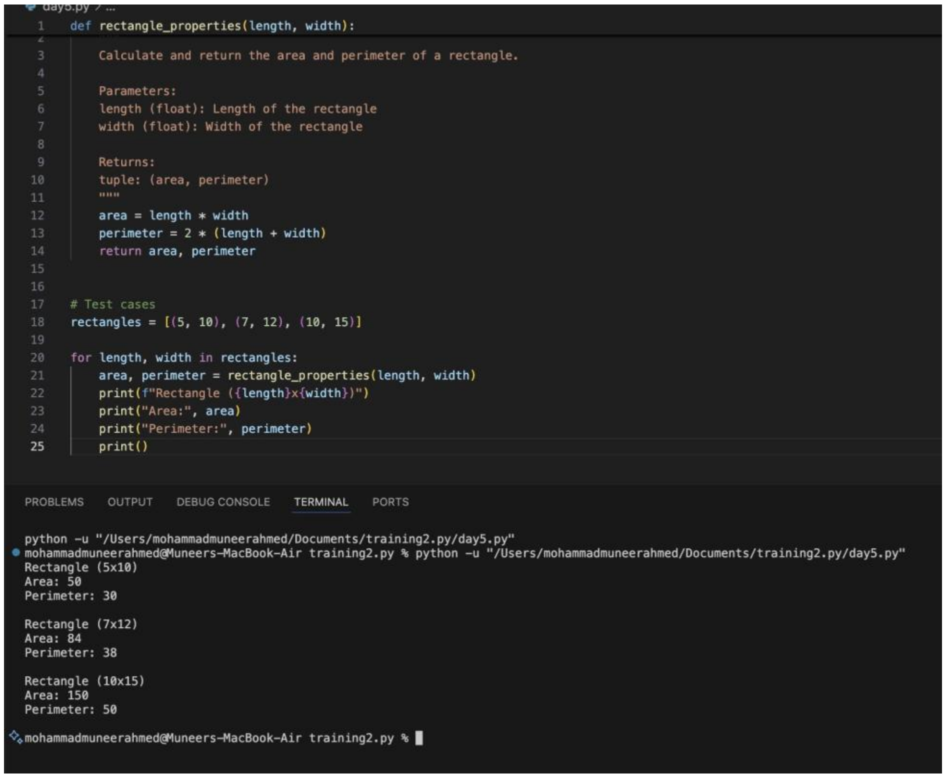
Lab 13: Code Refactoring Using AI Assistance

Improving Legacy Code for Readability, Maintainability, and Performance

Task 1: Refactoring – Removing Code Duplication

Scenario: In this task, AI assistance was used to refactor a legacy Python script containing repeated blocks of code that calculate the area and perimeter of rectangles. The goal was to eliminate duplication by introducing a reusable function with proper docstrings.

Prompt: # Refactor this legacy script by creating a reusable function to calculate area and perimeter of a rectangle with proper docstring Code:



```python
def rectangle_properties(length, width):

    Calculate and return the area and perimeter of a rectangle.

    Parameters:
    length (float): Length of the rectangle
    width (float): Width of the rectangle

    Returns:
    tuple: (area, perimeter)
    """
    area = length * width
    perimeter = 2 * (length + width)
    return area, perimeter


# Test cases
rectangles = [(5, 10), (7, 12), (10, 15)]

for length, width in rectangles:
    area, perimeter = rectangle_properties(length, width)
    print(f"Rectangle ({length}x{width})")
    print("Area:", area)
    print("Perimeter:", perimeter)
    print()
```

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Rectangle (5x10)
Area: 50
Perimeter: 30

Rectangle (7x12)
Area: 84
Perimeter: 38

Rectangle (10x15)
Area: 150
Perimeter: 50

mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

Rectangle (5x10) — Area: 50, Perimeter: 30

Rectangle (7x12) — Area: 84, Perimeter: 38

Rectangle (10x15) — Area: 150, Perimeter: 50

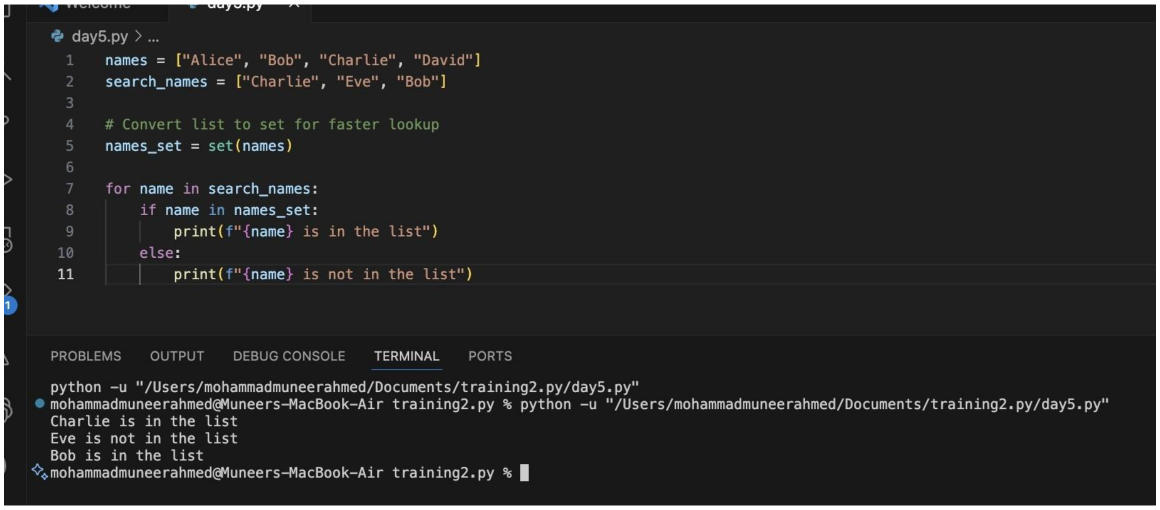
Observation:

The AI successfully eliminated all code duplication by extracting a reusable *rectangle_properties* function that returns a tuple of (area, perimeter). The refactored code uses a loop over test cases and includes a full docstring, improving both readability and maintainability. No logic was altered — all outputs match the original hardcoded results.

Task 2: Refactoring – Optimizing Loops and Conditionals

Scenario: This task involved using AI to analyze and refactor a script that used nested loops to check for element membership. The AI replaced the inefficient O(n²) nested loop with an optimized O(1) set lookup.

Prompt: # Refactor this nested loop to use a set for faster O(1) lookup when checking name membership Code:



day5.py > ...

```python
names = ["Alice", "Bob", "Charlie", "David"]
search_names = ["Charlie", "Eve", "Bob"]

# Convert list to set for faster lookup
names_set = set(names)

for name in search_names:
    if name in names_set:
        print(f"{name} is in the list")
    else:
        print(f"{name} is not in the list")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Charlie is in the list
Eve is not in the list
Bob is in the list
mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

Charlie is in the list

Eve is not in the list

Bob is in the list

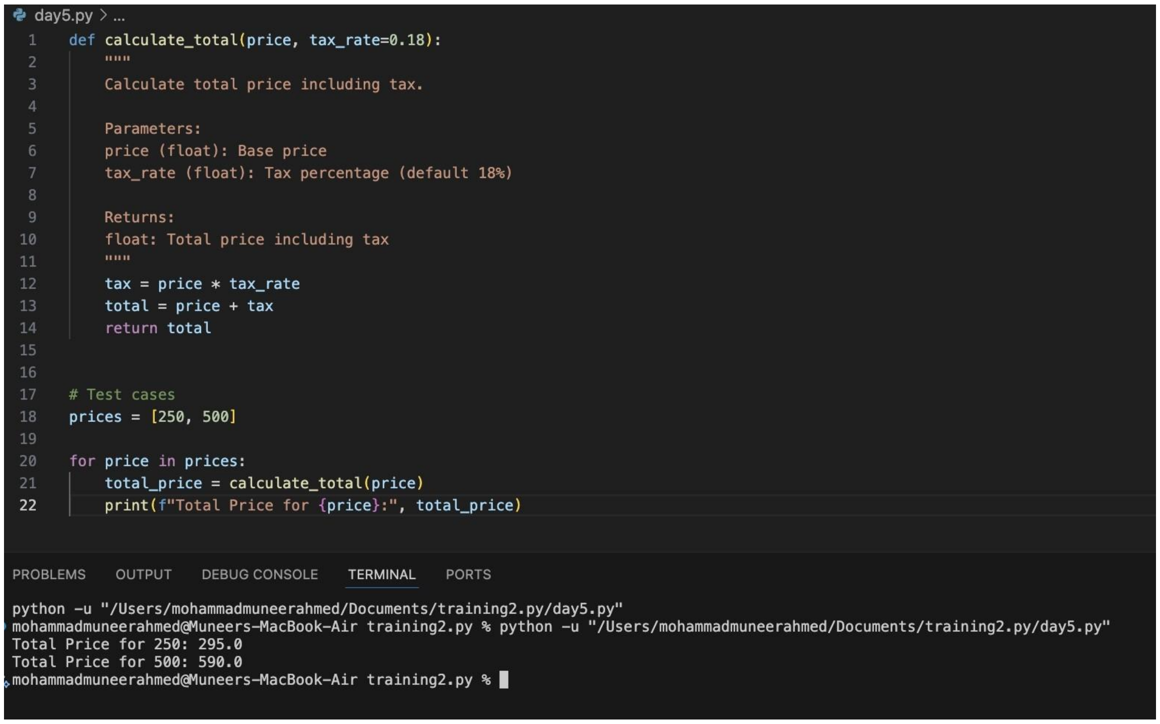
Observation:

The AI replaced the inefficient nested loop with a set conversion and O(1) membership check. This significantly improved performance for large lists, as set lookups are constant time. The output remains identical while the logic is cleaner and more Pythonic. This demonstrates how AI assistance can identify and fix performance bottlenecks effectively.

Task 3: Refactoring – Extracting Reusable Functions

Scenario: This task refactored a legacy script where price and tax calculations were written inline and repeated for each price value. The AI extracted the logic into a reusable *calculate_total* function with a default tax rate parameter.

Prompt: # Refactor repeated price+tax calculations into a reusable calculate_total(price, tax_rate=0.18) function with docstring Code:



day5.py > ...

```python
def calculate_total(price, tax_rate=0.18):
    """
    Calculate total price including tax.

    Parameters:
    price (float): Base price
    tax_rate (float): Tax percentage (default 18%)

    Returns:
    float: Total price including tax
    """
    tax = price * tax_rate
    total = price + tax
    return total


# Test cases
prices = [250, 500]

for price in prices:
    total_price = calculate_total(price)
    print(f"Total Price for {price}:", total_price)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Total Price for 250: 295.0
Total Price for 500: 590.0
mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

Total Price for 250: 295.0

Total Price for 500: 590.0

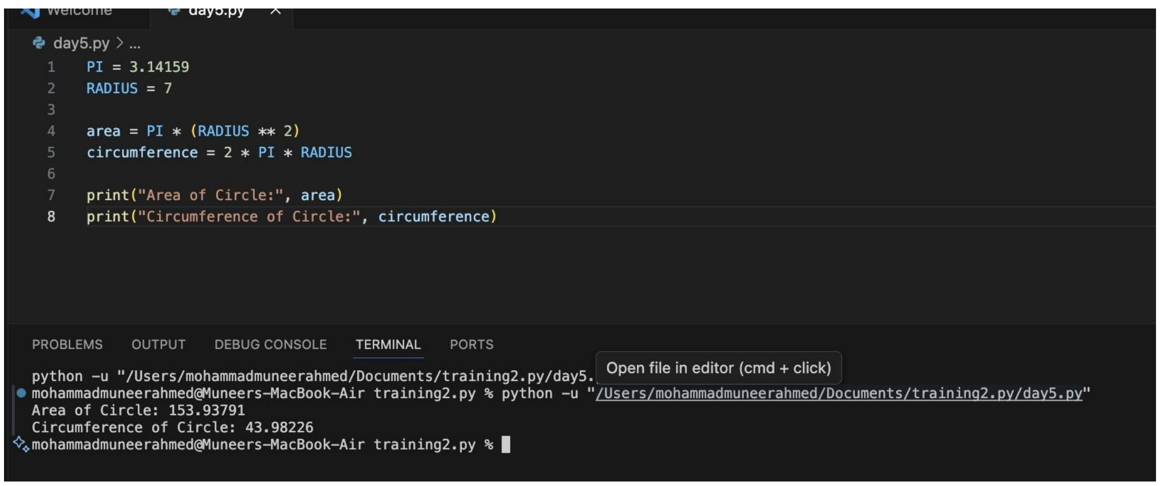
Observation:

The AI produced a clean, parameterized function with a default tax rate of 18%, making it flexible for other scenarios. The refactored version is much easier to maintain — changing the tax rate now requires only a single update to the function signature. This illustrates the DRY (Don't Repeat Yourself) principle applied through AI-assisted refactoring.

Task 4: Refactoring – Replacing Hardcoded Values with Constants

Scenario: This task involved identifying and replacing magic numbers in a circle calculation script with named constants. The AI introduced PI and RADIUS constants, improving the maintainability and readability of the code.

Prompt: # Replace hardcoded magic numbers in the circle calculations with named constants PI and RADIUS Code:


Welcome day5.py

day5.py > ...

```
PI = 3.14159
RADIUS = 7

area = PI * (RADIUS ** 2)
circumference = 2 * PI * RADIUS

print("Area of Circle:", area)
print("Circumference of Circle:", circumference)
```

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.
mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Area of Circle: 153.93791
Circumference of Circle: 43.98226
mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

Area of Circle: 153.93791

Circumference of Circle: 43.98226

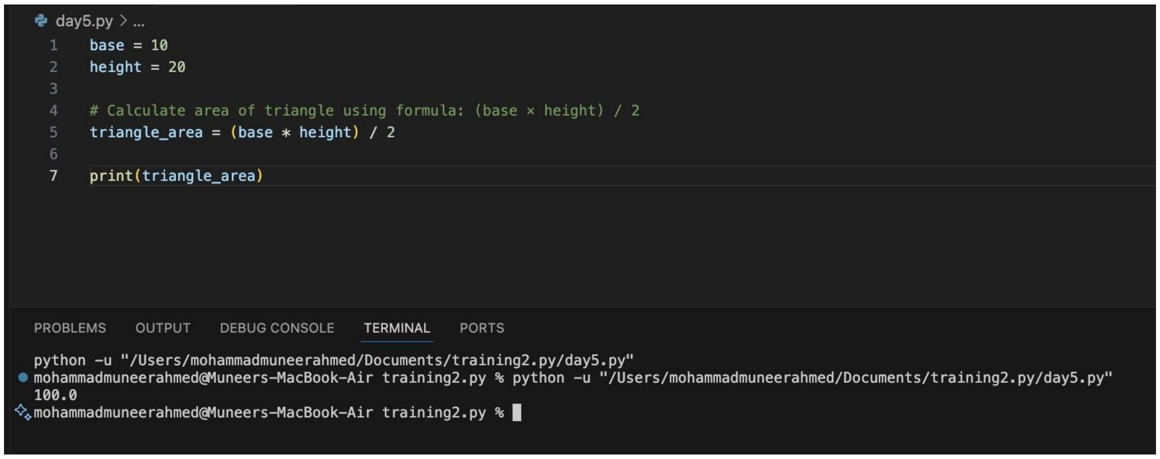
Observation:

The AI correctly identified the hardcoded values 3.14159 and 7 as magic numbers and replaced them with descriptive constants PI and RADIUS. The output is identical, but the code is now far easier to maintain — changing the radius requires only a single constant update. This approach is especially valuable in larger codebases where the same constant may appear in many places.

Task 5: Refactoring – Improving Variable Naming and Readability

Scenario: This task focused on improving the readability of a script that used single-character variable names (a, b, c) to calculate the area of a triangle. The AI renamed variables to be descriptive and added an inline comment explaining the formula.

Prompt: # Refactor this script with descriptive variable names and an inline comment explaining the triangle area formula Code:


day5.py > ...

```
base = 10
height = 20

# Calculate area of triangle using formula: (base × height) / 2
triangle_area = (base * height) / 2

print(triangle_area)
```

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
100.0
mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

```
100.0
```

Observation:

The AI renamed the cryptic variables a, b, c to base, height, and triangle_area, and added a comment explaining the geometric formula used. The output (100.0) is identical to the original. This demonstrates that AI-assisted refactoring can significantly improve code clarity without altering program behavior, making it much easier for other developers to understand and maintain the code.