AI Assisted Coding

Assignment – 13.3

M.Sathwik || 2303A51483 || Batch:- 08

Task 1: Refactoring – Removing Code Duplication

Objective

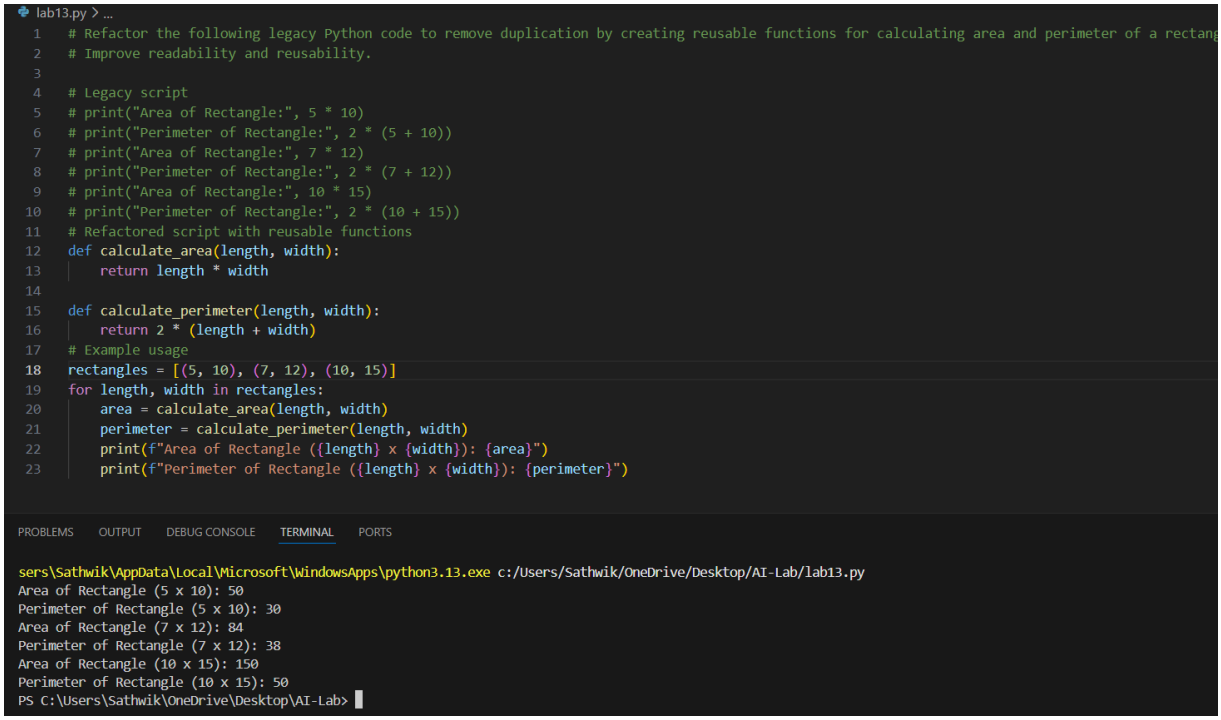
To eliminate repeated logic by extracting reusable functions.

Task Description

Use AI assistance to refactor a legacy Python script that contains repeated blocks of code calculating the area and perimeter of rectangles.

Starter (Legacy) Code

```
# Legacy script with repeated logic print("Area
of Rectangle:", 5 * 10) print("Perimeter of
Rectangle:", 2 * (5 + 10)) print("Area of
Rectangle:", 7 * 12) print("Perimeter of
Rectangle:", 2 * (7 + 12)) print("Area of
Rectangle:", 10 * 15) print("Perimeter of
Rectangle:", 2 * (10 + 15))
```



Task 2: Refactoring – Optimizing Loops and Conditionals

Objective

To improve performance by replacing inefficient nested loops with optimized structures.

Task Description

Use AI to analyze and refactor a script that checks the presence of elements using nested loops. Starter (Legacy) Code

names = ["Alice", "Bob", "Charlie", "David"]

search_names = ["Charlie", "Eve", "Bob"] for

s in search_names:

found = False

for n in

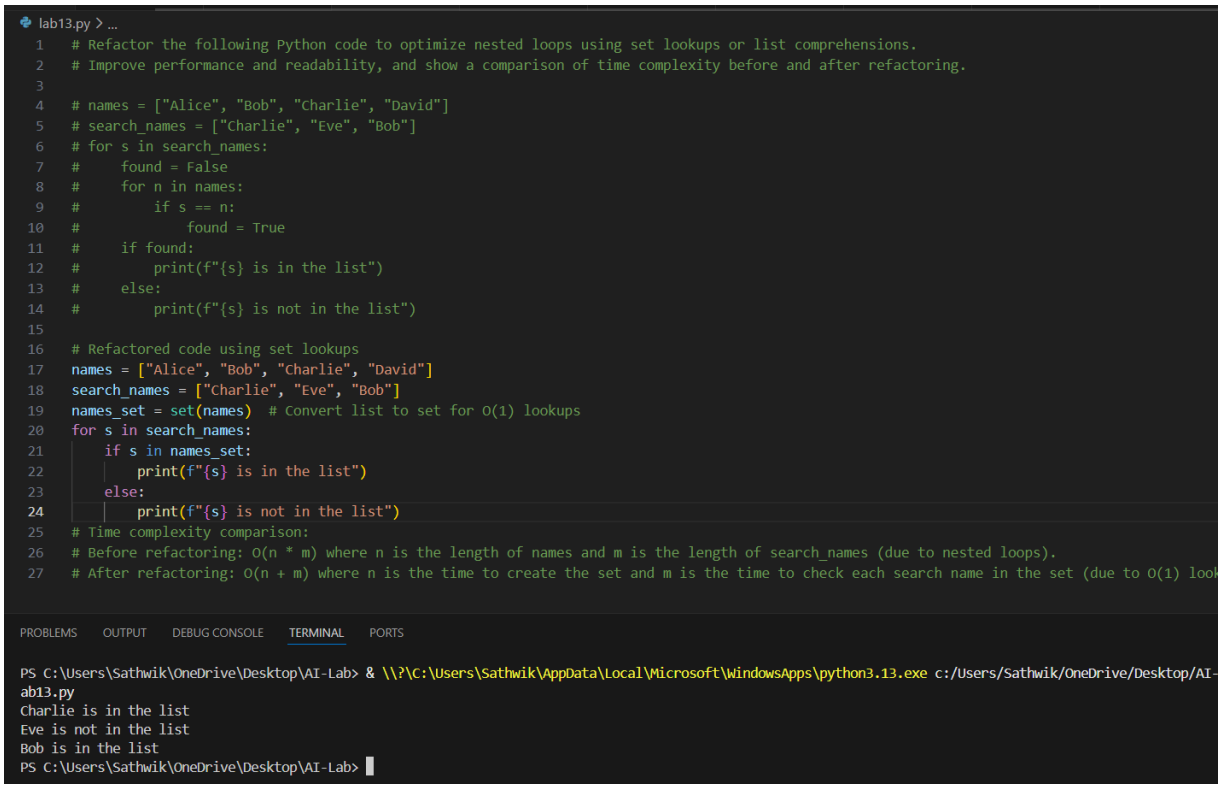
names: if s ==

n: found =

True if found:

print(f"{s} is in the list") else:

print(f"{s} is not in the list")



Task 3: Refactoring – Extracting Reusable Functions

Objective

To modularize code by extracting calculations into reusable functions.

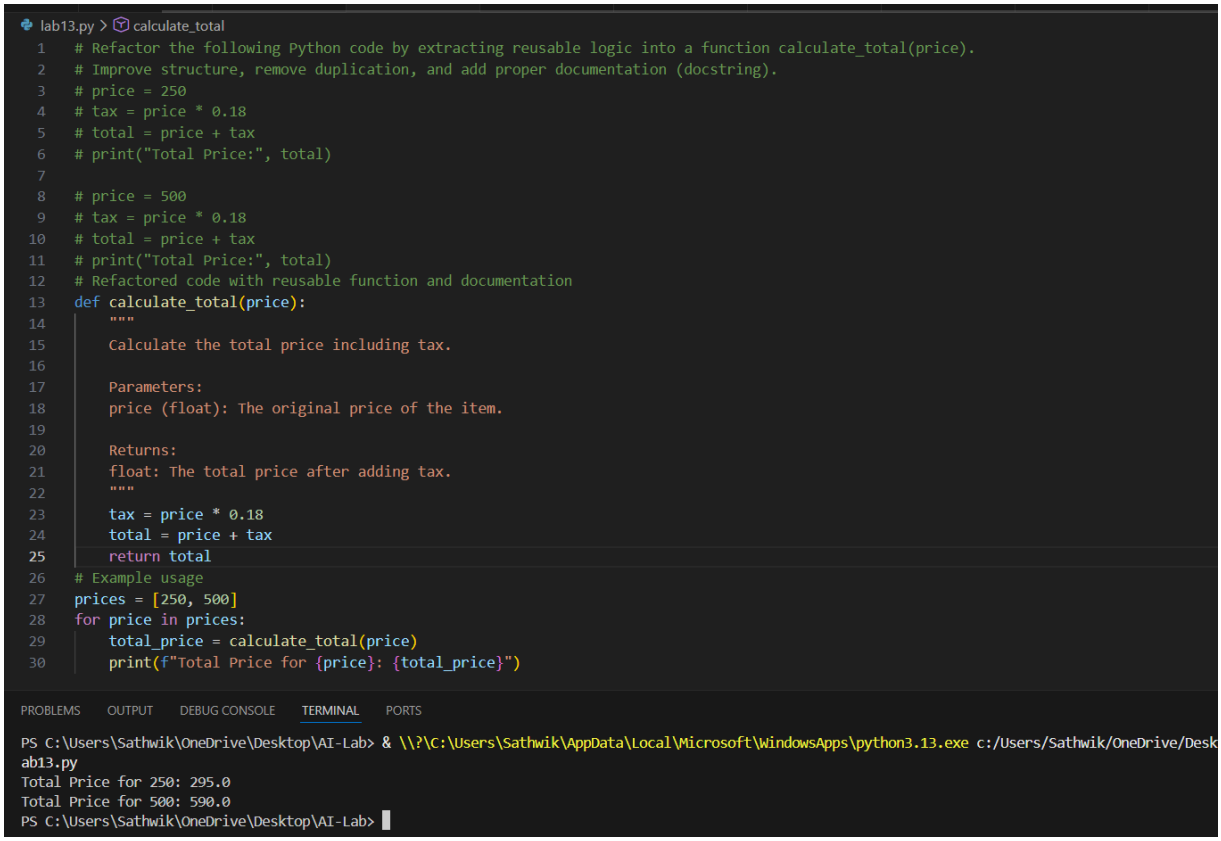
Task Description

Refactor a legacy script where price and tax calculations are written inline.

Starter (Legacy) Code price = 250 tax = price * 0.18 total = price + tax

print("Total Price:", total) price = 500 tax = price * 0.18 total = price + tax

print("Total Price:", total)



Task 4: Refactoring – Replacing Hardcoded Values with Constants

Objective

To improve maintainability by replacing magic numbers with named constants.

Task Description

Use AI to identify hardcoded values and replace them with constants. Starter

(Legacy) Code

print("Area of Circle:", 3.14159 * (7 ** 2)) print("Circumference

of Circle:", 2 * 3.14159 * 7)

```python
# Refactor the following Python code by replacing hardcoded values with named constants (e.g., PI, RADIUS).
# Improve readability and maintainability, and explain the benefits of using named constants in code.

# print("Area of Circle:", 3.14159 * (7 ** 2))
# print("Circumference of Circle:", 2 * 3.14159 * 7)
# Refactored code with named constants
PI = 3.14159
RADIUS = 7
area_of_circle = PI * (RADIUS ** 2)
circumference_of_circle = 2 * PI * RADIUS
print(f"Area of Circle: {area_of_circle}")
print(f"Circumference of Circle: {circumference_of_circle}")
# Benefits of using named constants:
# 1. Readability: Named constants make the code more readable and self-explanatory.
# It is easier to understand what PI and RADIUS represent compared to hardcoded values.
# 2. Maintainability: If the value of a constant needs to be changed (e.g., if we want to use a more precise value of PI),
# we only need to update it in one place, rather than searching through the code for all occurrences of the hardcoded value.
# 3. Avoiding Magic Numbers: Using named constants helps to avoid "magic numbers" in the code,
# which can be confusing and make the code less clear. It provides context to the values being used.
```

```
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab> & \\?\C:\Users\Sathwik\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/Sathwik/OneDrive/Des
Area of Circle: 153.93791
Circumference of Circle: 43.98226
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab>
```

Task 5: Refactoring – Improving Variable Naming and Readability

Objective

To enhance readability using descriptive variable names and comments.

Task Description

Refactor a script with unclear variable names.

Starter (Legacy) Code a = 10 b = 20 c = a * b

/ 2 print(c)

```python
# Refactor the following Python code by improving variable names and readability.
# Add descriptive names and inline comments while keeping the output the same, and explain the importance of meaningful variable names in code.

# a = 10
# b = 20
# c = a * b / 2
# print(c)
# Refactored code with improved variable names and inline comments
base_length = 10     # Length of the base of the triangle
height = 20          # Height of the triangle
area_of_triangle = (base_length * height) / 2   # Calculate the area of the triangle using the formula (base * height) / 2
print(f"Area of the triangle: {area_of_triangle}")   # Output the area of the triangle
# Importance of meaningful variable names:
# 1. Readability: Meaningful variable names make the code easier to read and understand.
# They provide context about what the variable represents, which helps other developers (or your future self)
# to quickly grasp the purpose of the variable without needing additional comments.
# 2. Maintainability: When variable names are descriptive, it is easier to maintain and update the code.
# Developers can quickly identify what each variable is used for, which reduces the chances of introducing bugs when making changes.
# 3. Collaboration: In a team environment, meaningful variable names facilitate better communication among developers.
# It allows team members to understand each other's code more easily, leading to more efficient collaboration and code reviews.
```

```
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab> & \\?\C:\Users\Sathwik\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/Sathwik/OneDrive/Desktop/AI-Lab/l
ab13.py
Area of the triangle: 100.0
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab>
```