AI ASSISTED CODING

LAB-13.1

Aashritha.

2303A51756

Batch-11

Task Description #1 (Refactoring – Removing Code Duplication)

• Task: Use AI to refactor a given Python script that contains multiple repeated code blocks.

• Instructions:

o Prompt AI to identify duplicate logic and replace it with functions or classes.

o Ensure the refactored code maintains the same output.

o Add docstrings to all functions.

• Sample 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))
```

• Expected Output:

o Refactored code with a reusable function and no duplication.

PROMPT:

#Identify duplicate logic and replace it with functions or classes, make sure the refactored code maintains the same output.

GIVEN CODE AND OUTPUT:

```python
#Identify duplicate logic and replace it with functions or classes, make sure the refactored code mainta
def calculate_rectangle_area(length, width):
    """Calculate the area of a rectangle given its length and width.
    input: length and width of the rectangle
    output: area of the rectangle"""
    return length * width

def calculate_rectangle_perimeter(length, width):
    """Calculate the perimeter of a rectangle given its length and width.
    input: length and width of the rectangle
    output: perimeter of the rectangle"""
    return 2 * (length + width)

print("Area of Rectangle:", calculate_rectangle_area(5, 10))
print("Perimeter of Rectangle:", calculate_rectangle_perimeter(5, 10))
print("Area of Rectangle:", calculate_rectangle_area(7, 12))
print("Perimeter of Rectangle:", calculate_rectangle_perimeter(7, 12))
print("Area of Rectangle:", calculate_rectangle_area(10, 15))
print("Perimeter of Rectangle:", calculate_rectangle_perimeter(10, 15))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS POSTMAN CONSOLE

```
PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/python.exe c:/Users/katta/OneDrive/Desktop/AIAC/la
Area of Rectangle: 50
Perimeter of Rectangle: 30
Area of Rectangle: 84
Perimeter of Rectangle: 38
Area of Rectangle: 150
Perimeter of Rectangle: 50
```

Task Description #2 (Refactoring – Optimizing Loops and Conditionals)

• Task: Use AI to analyze a Python script with nested loops

and complex conditionals.

• Instructions:

o Ask AI to suggest algorithmic improvements (e.g.,

replace nested loops with set lookups or

comprehensions).

o Implement changes while keeping logic intact.

o Compare execution time before and after refactoring.

• Sample Legacy Code:

 # Legacy inefficient 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")

- **Expected Output:**

o Optimized code using set lookups with performance

comparison table.

PROMPT:

#Refactor this Python code to optimize nested loops: replace inner loops with set lookups or comprehensions while keeping the logic identical. Include a simple timing comparison before and after refactoring.

GIVEN CODE AND OUTPUT:

```
#Refactor this Python code to optimize nested loops: replace inner loops with set lookups or compr
"""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
            break
    if found:
        print(f"{s} is in the list")
else:
    print(f"{s} is not in the list")"""
import time
names = ["Alice", "Bob", "Charlie", "David"]
search_names = ["Charlie", "Eve", "Bob"]
# Timing before refactoring
start_time = time.time()
for s in search_names:
    found = False
    for n in names:
        if s == n:
            found = True
            break
    if found:
        print(f"{s} is in the list")
    else:
        print(f"{s} is not in the list")
end_time = time.time()
print(f"Time taken before refactoring: {end_time - start_time:.6f} seconds")
# Refactored code using set lookups
start_time = time.time()
names_set = set(names)  # Convert list to set for O(1) lookups
for s in search_names:
    if s in names_set:
        print(f"{s} is in the list")
    else:
        print(f"{s} is not in the list")
end_time = time.time()
print(f"Time taken after refactoring: {end_time - start_time:.6f} seconds")
```

```
PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katt
Charlie is in the list
Eve is not in the list
Bob is in the list
Time taken before refactoring: 0.001000 seconds
Charlie is in the list
Eve is not in the list
Bob is in the list
Time taken after refactoring: 0.000000 seconds
PS C:\Users\katta\OneDrive\Desktop\AIAC>
```

Task Description #3 (Refactoring – Extracting Reusable Functions)

• Task: Use AI to refactor a legacy script where multiple calculations are embedded directly inside the main code block.

• Instructions:

o Identify repeated or related logic and extract it into reusable functions.

o Ensure the refactored code is modular, easy to read, and documented with docstrings.

• Sample Legacy Code:

```
# Legacy script with inline repeated logic
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)
```

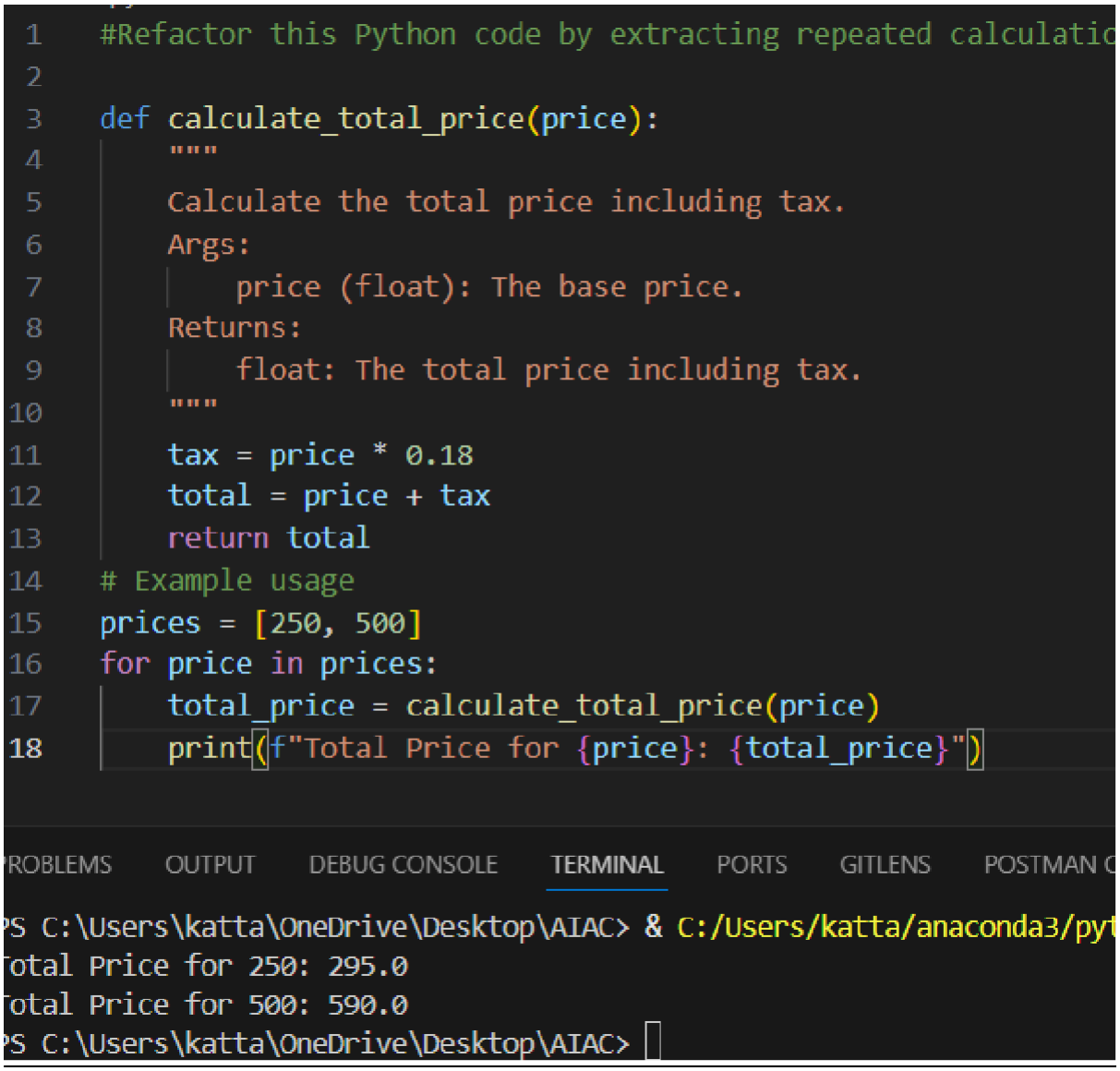
• Expected Output:

o Code with a function calculate_total(price) that can be reused for multiple price inputs.

PROMPT:

Refactor this Python code by extracting repeated calculations into a reusable function with docstrings. Make the code modular, readable, and allow multiple inputs without duplicating logic.

GIVEN CODE AND OUTPUT:



```python
#Refactor this Python code by extracting repeated calculatic

def calculate_total_price(price):
    """
    Calculate the total price including tax.
    Args:
        price (float): The base price.
    Returns:
        float: The total price including tax.
    """
    tax = price * 0.18
    total = price + tax
    return total
# Example usage
prices = [250, 500]
for price in prices:
    total_price = calculate_total_price(price)
    print(f"Total Price for {price}: {total_price}")
```

```
ROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  POSTMAN C
S C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/pyt
otal Price for 250: 295.0
otal Price for 500: 590.0
S C:\Users\katta\OneDrive\Desktop\AIAC>
```

Task Description #4 (Refactoring – Replacing Hardcoded Values with Constants)

• Task: Use AI to identify and replace all hardcoded “magic numbers” in the code with named constants.

• Instructions:

o Create constants at the top of the file.

o Replace all hardcoded occurrences in calculations with these constants.

o Ensure the code remains functional and is easier to maintain.

• Sample Legacy Code:

 # Legacy script with hardcoded values

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

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

• **Expected Output:**

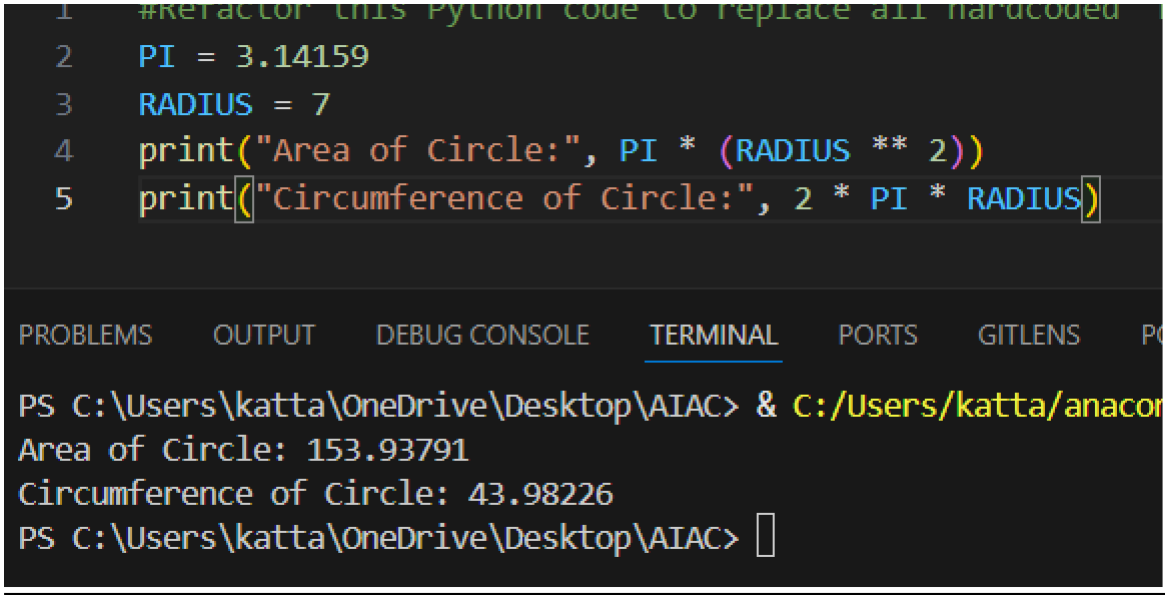
o Code with constants like PI = 3.14159 and RADIUS

= 7 used in calculations.

PROMPT:

Refactor this Python code to replace all hardcoded “magic numbers” with named constants at the top of the file. Ensure calculations remain correct and the code is easier to maintain.

GIVEN CODE AND OUTPUT:



Task Description #5 (Refactoring – Improving Variable Naming and Readability)

• **Task: Use AI to improve readability by renaming unclear**

variables and adding inline comments.

• **Instructions:**

o Replace vague names with meaningful ones.

o Add comments where logic is not obvious.

o Keep functionality exactly the same.

• **Sample Legacy Code:**

 # Legacy script with poor variable names

a = 10

b = 20

c = a * b / 2

print(c)

• **Expected Output:**

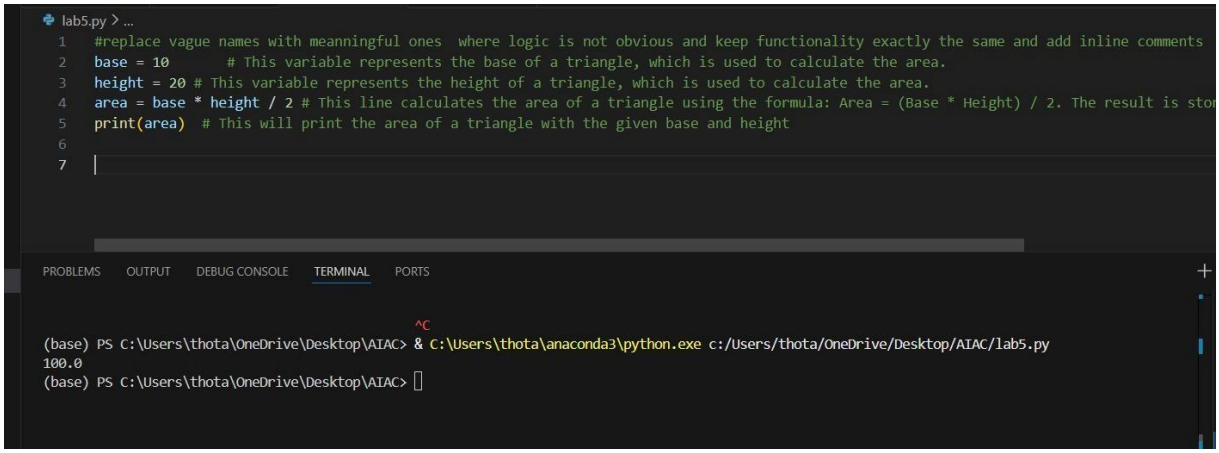
o Code with descriptive variable names like base,

height, area_of_triangle, plus explanatory comments.

PROMPT:

#replace vague names with meanningful ones where logic is not obvious and keep functionality exactly the same and add inline comments

GIVEN CODE AND OUTPUT:



Task Description #6 (Refactoring – Removing Redundant Conditional Logic)

• **Task: Use AI to refactor a Python script that contains repeated if–else**

logic for grading students.

• **Instructions:**

o Ask AI to identify redundant conditional checks.

o Replace them with a reusable function.

o Ensure grading logic remains unchanged.

• **Code:**

marks = 85

if marks >= 90:

print("Grade A")

elif marks >= 75:

print("Grade B")

else:

print("Grade C")

marks = 72

if marks >= 90:

print("Grade A")

elif marks >= 75:

print("Grade B")

else:

print("Grade C")

- **Expected Output:**

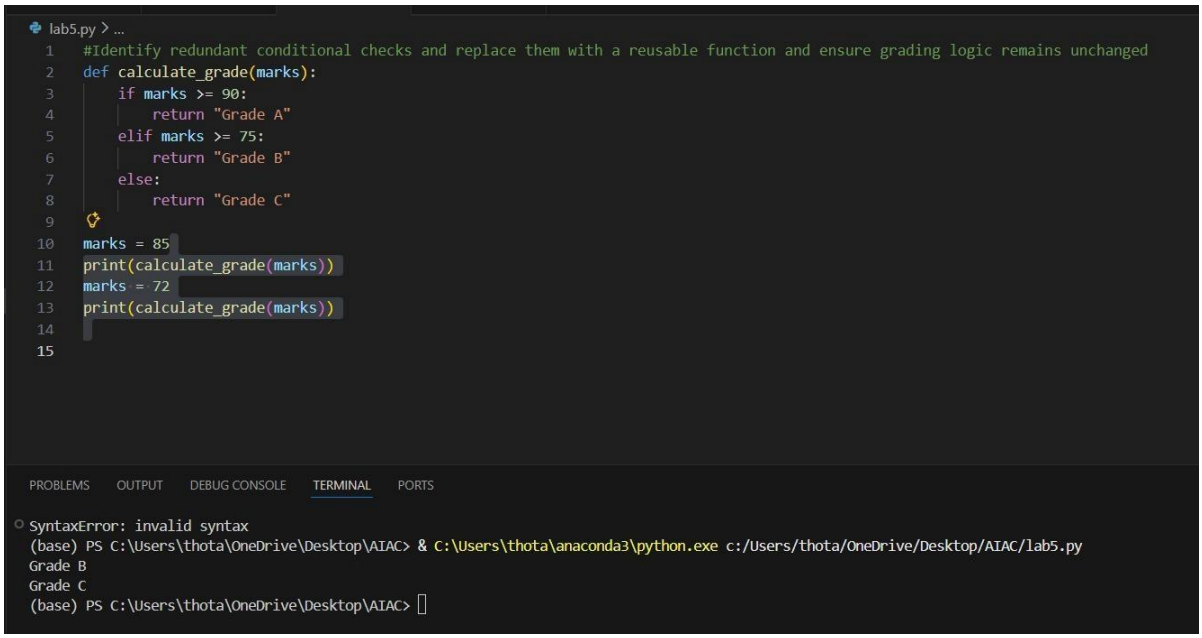
o A reusable function like calculate_grade(marks) with clean logic and

docstring.

PROMPT:

#Identify redundant conditional checks and replace them with a reusable function and ensure grading logic remains unchanged

GIVEN CODE AND OUTPUT:



Task Description #7 (Refactoring – Converting Procedural Code to Functions)

- **Task: Use AI to refactor procedural input–processing logic into**

functions.

- **Instructions:**

o Identify input, processing, and output sections.

o Convert each into a separate function.

o Improve code readability without changing behavior.

• Sample Legacy Code:

num = int(input("Enter number: "))

square = num * num

print("Square:", square)

• Expected Output:

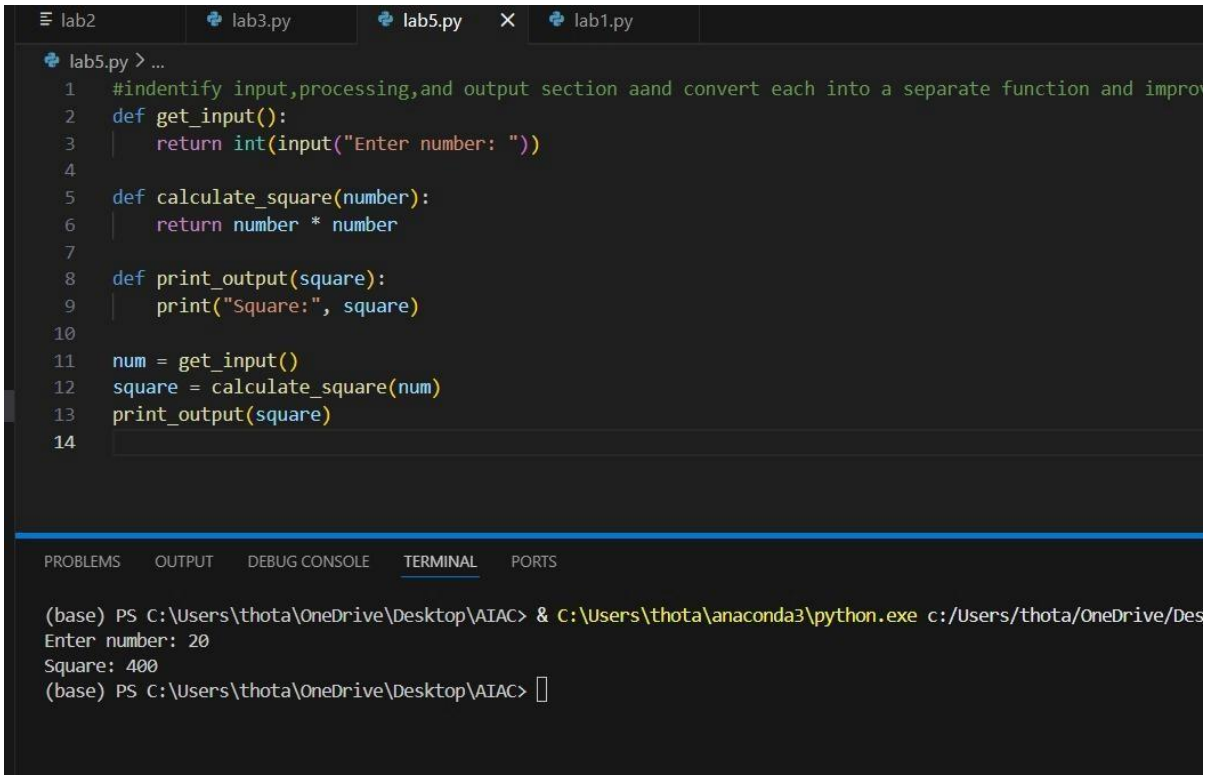
o Modular code using functions like get_input(), calculate_square(),

and display_result().

PROMPT:

#indentify input, processing, and output section and convert each into a separate function and improve code readability without changing behaviour.

GIVEN CODE AND OUTPUT:



Task Description #8 (Refactoring – Optimizing List Processing)

• Task: Use AI to refactor inefficient list processing logic.

• Instructions:

o Ask AI to replace manual loops with list comprehensions or built-in

functions.

o Ensure output remains identical.

• Sample Legacy Code:

nums = [1, 2, 3, 4, 5]

squares = []

for n in nums:

squares.append(n * n)

print(squares)

• Expected Output:

o Optimized version using list comprehension with improved

readability

PROMPT:

Refactor this Python code to use list comprehensions or built-in functions instead of manual loops for list processing. Keep the output identical but make the code more concise and readable.

GIVEN CODE AND OUTPUT: