

# **AI ASSISTED CODING**

## **LAB-13.1**

**M.ABHISHEK**

**2303A51762**

**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:**

```
1 #Identify duplicate logic and replace it with functions or classes, make sure the refactored code mainta
2 def calculate_rectangle_area(length, width):
3     """Calculate the area of a rectangle given its length and width.
4     input: length and width of the rectangle
5     output: area of the rectangle"""
6     return length * width
7
8 def calculate_rectangle_perimeter(length, width):
9     """Calculate the perimeter of a rectangle given its length and width.
10    input: length and width of the rectangle
11    output: perimeter of the rectangle"""
12    return 2 * (length + width)
13
14    print("Area of Rectangle:", calculate_rectangle_area(5, 10))
15    print("Perimeter of Rectangle:", calculate_rectangle_perimeter(5, 10))
16    print("Area of Rectangle:", calculate_rectangle_area(7, 12))
17    print("Perimeter of Rectangle:", calculate_rectangle_perimeter(7, 12))
18    print("Area of Rectangle:", calculate_rectangle_area(10, 15))
19    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 comprehensions
"""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:**

```
1 #Refactor this Python code by extracting repeated calculation
2
3 def calculate_total_price(price):
4     """
5     Calculate the total price including tax.
6     Args:
7         price (float): The base price.
8     Returns:
9         float: The total price including tax.
10    """
11    tax = price * 0.18
12    total = price + tax
13    return total
14
15 # Example usage
16 prices = [250, 500]
17 for price in prices:
18     total_price = calculate_total_price(price)
19     print(f"Total Price for {price}: {total_price}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS POSTMAN C

```
PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/pyt
Total Price for 250: 295.0
Total Price for 500: 590.0
PS 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))
```

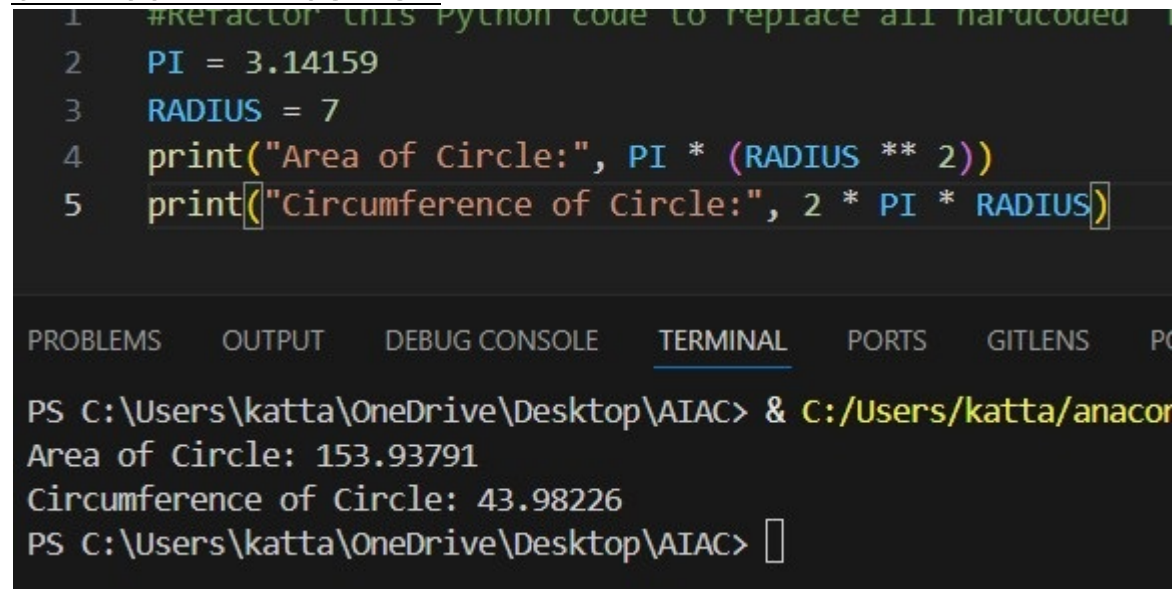
```
print("Circumference of Circle:", 2 *  
3.14159 * 7)
```

- Expected Output: o Code with constants like  $\text{PI} = 3.14159$  and  $\text{RADIUS} = 7$  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:**



The screenshot shows a code editor with the following Python code:

```
1 #Refactor this Python code to replace all hardcoded  
2 PI = 3.14159  
3 RADIUS = 7  
4 print("Area of Circle:", PI * (RADIUS ** 2))  
5 print("Circumference of Circle:", 2 * PI * RADIUS)
```

Below the code editor, the terminal output is displayed:

```
PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/python.exe C:/Users/katta/OneDrive/Desktop/AIAC/area_circle.py  
Area of Circle: 153.93791  
Circumference of Circle: 43.98226  
PS C:\Users\katta\OneDrive\Desktop\AIAC>
```

#### **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
```

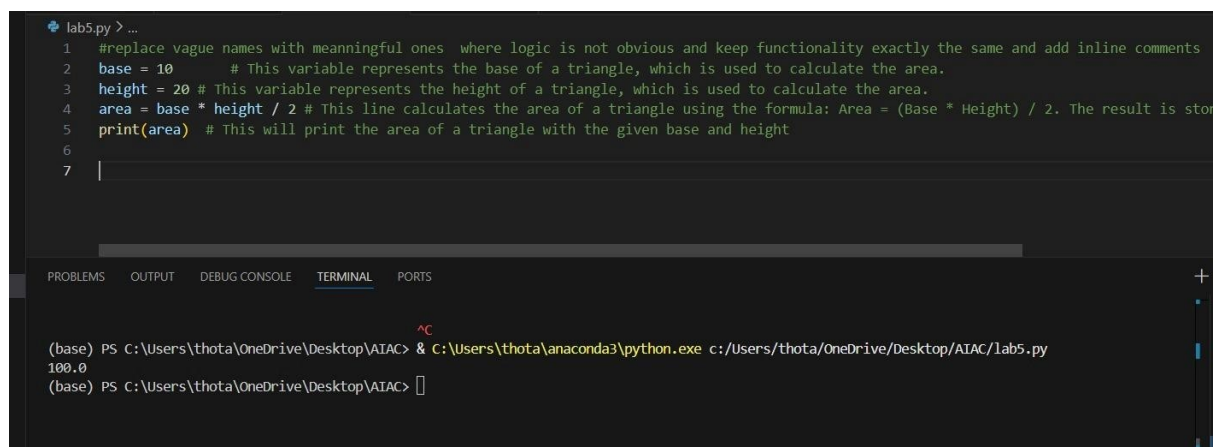
```
print(c)
```

- Expected Output: o Code with descriptive variable names like base, height, area\_of\_triangle, plus explanatory comments.

### **PROMPT:**

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

### **CODE AND OUTPUT:**



The screenshot shows a code editor with a file named 'lab5.py'. The code is as follows:

```
1 #replace vague names with meaningful ones where logic is not obvious and keep functionality exactly the same and add inline comments
2 base = 10 # This variable represents the base of a triangle, which is used to calculate the area.
3 height = 20 # This variable represents the height of a triangle, which is used to calculate the area.
4 area = base * height / 2 # This line calculates the area of a triangle using the formula: Area = (Base * Height) / 2. The result is stored in the variable 'area'.
5 print(area) # This will print the area of a triangle with the given base and height
6
7
```

Below the code editor is a terminal window. It shows the command to run the script and the output:

```
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:\Users\thota\anaconda3\python.exe c:/Users/thota/OneDrive/Desktop/AIAC/lab5.py
100.0
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

### **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:

```
lab5.py > ...
1  #Identify redundant conditional checks and replace them with a reusable function and ensure grading logic remains unchanged
2  def calculate_grade(marks):
3      if marks >= 90:
4          return "Grade A"
5      elif marks >= 75:
6          return "Grade B"
7      else:
8          return "Grade C"
9
10 marks = 85
11 print(calculate_grade(marks))
12 marks = 72
13 print(calculate_grade(marks))
14
15
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

○ SyntaxError: invalid syntax  
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:\Users\thota\anaconda3\python.exe c:/Users/thota/OneDrive/Desktop/AIAC/lab5.py  
Grade B  
Grade C  
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC>



## 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)
```

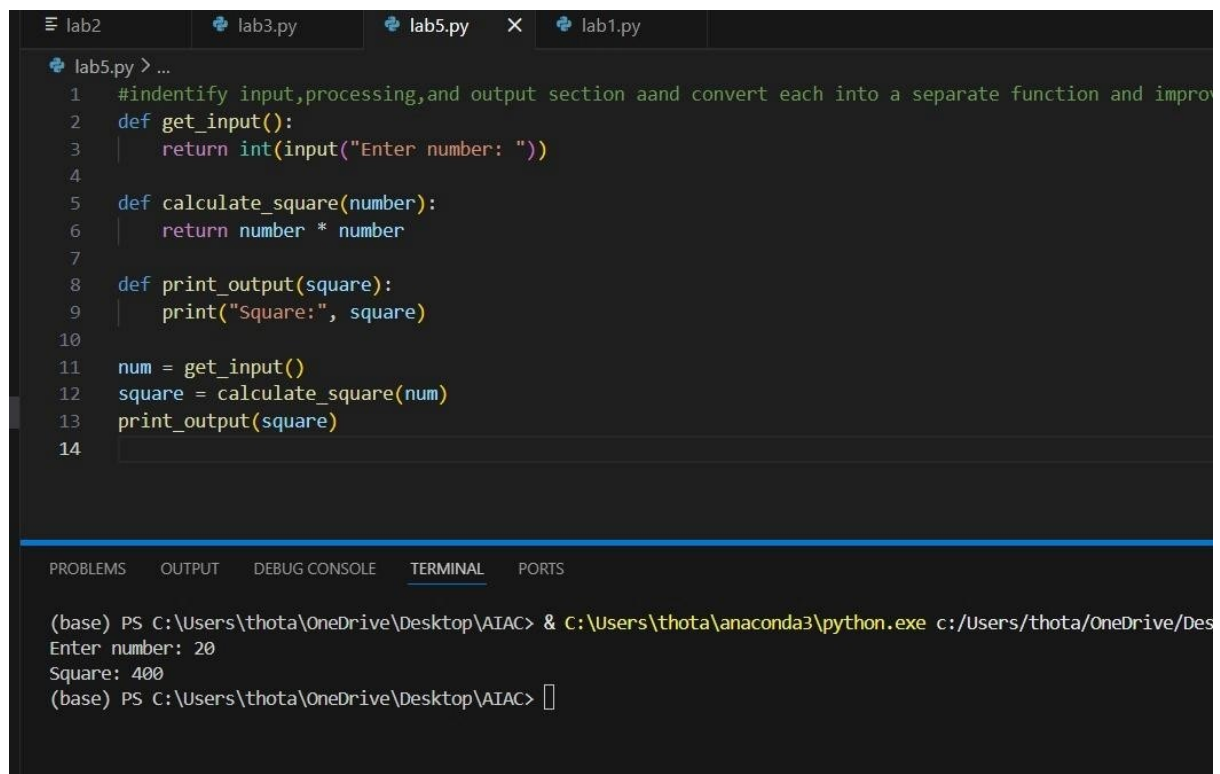
- Expected Output:

- o Modular code using functions like `get_input()`, `calculate_square()`, and `display_result()`.

### PROMPT:

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

### GIVEN CODE AND OUTPUT:



The screenshot shows a code editor with four tabs: lab2, lab3.py, lab5.py, and lab1.py. The active tab is lab5.py, which contains the following Python code:

```
1 #identify input,processing,and output section aand convert each into a separate function and improv
2 def get_input():
3     return int(input("Enter number: "))
4
5 def calculate_square(number):
6     return number * number
7
8 def print_output(square):
9     print("Square:", square)
10
11 num = get_input()
12 square = calculate_square(num)
13 print_output(square)
14
```

Below the code editor is a terminal window with the following output:

```
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:\Users\thota\anaconda3\python.exe c:/Users/thota/OneDrive/Des
Enter number: 20
Square: 400
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC> █
```

## 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)`

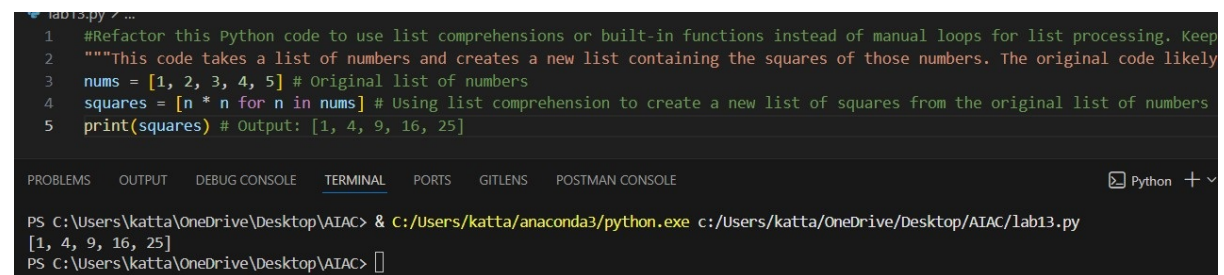
`print(squares)` • Expected Output: o Optimized

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:



```
1 #Refactor this Python code to use list comprehensions or built-in functions instead of manual loops for list processing. Keep
2 """This code takes a list of numbers and creates a new list containing the squares of those numbers. The original code likely
3 nums = [1, 2, 3, 4, 5] # Original list of numbers
4 squares = [n * n for n in nums] # Using list comprehension to create a new list of squares from the original list of numbers
5 print(squares) # Output: [1, 4, 9, 16, 25]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS POSTMAN CONSOLE Python + v

```
PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/python.exe c:/Users/katta/OneDrive/Desktop/AIAC/lab13.py
[1, 4, 9, 16, 25]
PS C:\Users\katta\OneDrive\Desktop\AIAC>
```