

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))
```

```
lab13.py > ...
1 # Refactor the following legacy Python code to remove duplication by creating reusable functions for calculating area and perimeter of a rectangle.
2 # Improve readability and reusability.
3
4 # Legacy script
5 # print("Area of Rectangle:", 5 * 10)
6 # print("Perimeter of Rectangle:", 2 * (5 + 10))
7 # print("Area of Rectangle:", 7 * 12)
8 # print("Perimeter of Rectangle:", 2 * (7 + 12))
9 # print("Area of Rectangle:", 10 * 15)
10 # print("Perimeter of Rectangle:", 2 * (10 + 15))
11 # Refactored script with reusable functions
12 def calculate_area(length, width):
13     return length * width
14
15 def calculate_perimeter(length, width):
16     return 2 * (length + width)
17 # Example usage
18 rectangles = [(5, 10), (7, 12), (10, 15)]
19 for length, width in rectangles:
20     area = calculate_area(length, width)
21     perimeter = calculate_perimeter(length, width)
22     print(f"Area of Rectangle ({length} x {width}): {area}")
23     print(f"Perimeter of Rectangle ({length} x {width}): {perimeter}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

sers\Sathwik\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/Sathwik/OneDrive/Desktop/AI-Lab/lab13.py
Area of Rectangle (5 x 10): 50
Perimeter of Rectangle (5 x 10): 30
Area of Rectangle (7 x 12): 84
Perimeter of Rectangle (7 x 12): 38
Area of Rectangle (10 x 15): 150
Perimeter of Rectangle (10 x 15): 50
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab>
```

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')
```

```
lab13.py > ...
1 # Refactor the following Python code to optimize nested loops using set lookups or list comprehensions.
2 # Improve performance and readability, and show a comparison of time complexity before and after refactoring.
3
4 # names = ["Alice", "Bob", "Charlie", "David"]
5 # search_names = ["Charlie", "Eve", "Bob"]
6 # for s in search_names:
7 #     found = False
8 #     for n in names:
9 #         if s == n:
10 #             found = True
11 #     if found:
12 #         print(f'{s} is in the list')
13 #     else:
14 #         print(f'{s} is not in the list')
15
16 # Refactored code using set lookups
17 names = ["Alice", "Bob", "Charlie", "David"]
18 search_names = ["Charlie", "Eve", "Bob"]
19 names_set = set(names) # Convert list to set for O(1) lookups
20 for s in search_names:
21     if s in names_set:
22         print(f'{s} is in the list')
23     else:
24         print(f'{s} is not in the list')
25
26 # Time complexity comparison:
27 # Before refactoring: O(n * m) where n is the length of names and m is the length of search_names (due to nested loops).
28 # After refactoring: O(n + m) where n is the time to create the set and m is the time to check each search name in the set (due to O(1) lookups).
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
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/lab13.py
Charlie is in the list
Eve is not in the list
Bob is in the list
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab>
```

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)

```
lab13.py > calculate_total
1  # Refactor the following Python code by extracting reusable logic into a function calculate_total(price).
2  # Improve structure, remove duplication, and add proper documentation (docstring).
3  # price = 250
4  # tax = price * 0.18
5  # total = price + tax
6  # print("Total Price:", total)
7
8  # price = 500
9  # tax = price * 0.18
10 # total = price + tax
11 # print("Total Price:", total)
12 # Refactored code with reusable function and documentation
13 def calculate_total(price):
14     """
15     Calculate the total price including tax.
16
17     Parameters:
18     price (float): The original price of the item.
19
20     Returns:
21     float: The total price after adding tax.
22     """
23     tax = price * 0.18
24     total = price + tax
25     return total
26
27 # Example usage
28 prices = [250, 500]
29 for price in prices:
30     total_price = calculate_total(price)
31     print(f"Total Price for {price}: {total_price}")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab> & \\?C:\Users\Sathwik\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/Sathwik/OneDrive/Desktop/ab13.py
Total Price for 250: 295.0
Total Price for 500: 590.0
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab> |
```

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)
```

```
lab13.py > ...
1 # Refactor the following Python code by replacing hardcoded values with named constants (e.g., PI, RADIUS).
2 # Improve readability and maintainability, and explain the benefits of using named constants in code.
3
4 # print("Area of Circle:", 3.14159 * (7 ** 2))
5 # print("Circumference of Circle:", 2 * 3.14159 * 7)
6 # Refactored code with named constants
7 PI = 3.14159
8 RADIUS = 7
9 area_of_circle = PI * (RADIUS ** 2)
10 circumference_of_circle = 2 * PI * RADIUS
11 print(f"Area of Circle: {area_of_circle}")
12 print(f"Circumference of Circle: {circumference_of_circle}")
13 # Benefits of using named constants:
14 # 1. Readability: Named constants make the code more readable and self-explanatory.
15 # It is easier to understand what PI and RADIUS represent compared to hardcoded values.
16 # 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),
17 # we only need to update it in one place, rather than searching through the code for all occurrences of the hardcoded value.
18 # 3. Avoiding Magic Numbers: Using named constants helps to avoid "magic numbers" in the code,
19 # which can be confusing and make the code less clear. It provides context to the values being used.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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 $a = 10$ $b = 20$ $c = a * b$

$/ 2$ $\text{print}(c)$

```
lab13.py > ...
1 # Refactor the following Python code by improving variable names and readability.
2 # Add descriptive names and inline comments while keeping the output the same, and explain the importance of meaningful variable names in code.
3
4 # a = 10
5 # b = 20
6 # c = a * b / 2
7 # print(c)
8 # Refactored code with improved variable names and inline comments
9 base_length = 10 # Length of the base of the triangle
10 height = 20 # Height of the triangle
11 area_of_triangle = (base_length * height) / 2 # Calculate the area of the triangle using the formula (base * height) / 2
12 print(f"Area of the triangle: {area_of_triangle}") # Output the area of the triangle
13 # Importance of meaningful variable names:
14 # 1. Readability: Meaningful variable names make the code easier to read and understand.
15 # They provide context about what the variable represents, which helps other developers (or your future self)
16 # to quickly grasp the purpose of the variable without needing additional comments.
17 # 2. Maintainability: When variable names are descriptive, it is easier to maintain and update the code.
18 # Developers can quickly identify what each variable is used for, which reduces the chances of introducing bugs when making changes.
19 # 3. Collaboration: In a team environment, meaningful variable names facilitate better communication among developers.
20 # It allows team members to understand each other's code more easily, leading to more efficient collaboration and code reviews.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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/1
ab13.py
Area of the triangle: 100.0
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab> |
```