

School of Computer Science and Artificial Intelligence

Lab Assignment # 2.5

Program : B. Tech (CSE)

Specialization :AIML

Course Title : AI Assisted Coding

Course Code : 23CS002PC304

Semester : VI

Academic Session : 2025-2026

Name of Student : k.sathwika

Enrollment No. : 2303A52096

Batch No. : 33

Date :09/01/26

TASK_01: Refactoring Odd/Even Logic (List Version)

Scenario:

You are improving legacy code.

Task Description:

Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.

The screenshot shows a dark-themed AI interface. At the top, there's a header bar with 'CHAT' and various icons. Below it, a title bar says '← SUM OF ODD AND EVEN NUMBERS PROGRAM REFA...'. A large text input area contains the instruction: 'Write a Python program to calculate the sum of odd and even numbers in a list.' Below this, a list of steps is shown with checkmarks: '✓ Read main.py file'. The text 'The file is empty. Let me create a clean, simple program to calculate the sum of odd and even numbers in a list:' follows. Another step '✓ Created + main.py' is listed. The text 'Done! I've created a Python program that:' is followed by a bulleted list: '• Calculates the sum of odd and even numbers in a list', '• Uses a simple, clear function with proper documentation', and '• Includes test code that demonstrates how to use it'. Below this, the text 'The program will output the sum of odd numbers, sum of even numbers, and the total sum for a given list.' is displayed. At the bottom, there are interaction icons (comment, share, like, dislike) and a credit 'Claude Haiku 4.5 • 1x'. A smaller window at the bottom asks 'Describe what to build next' with dropdowns for 'Agent' and 'Auto' and a 'Υ' icon. The footer includes links for 'Python', '3.11.9 (Microsoft Store)', 'Go Live', and a bell icon.

The screenshot shows a code editor window with a dark theme. The file is named 'main.py'. The code defines a function 'calculate_sums' that takes a list of integers and returns a tuple of odd and even sums. It includes docstrings for arguments and return values. The code then tests this function with a list of numbers from 1 to 10 and prints the results.

```
Assignment 1.2 > Assignment 2.5 > main.py > ...
1  def calculate_sums(numbers):
2      """Calculate the sum of odd and even numbers in a list.
3
4      Args:
5          numbers: A list of integers
6
7      Returns:
8          A tuple of (odd_sum, even_sum)
9      """
10     odd_sum = 0
11     even_sum = 0
12
13     for num in numbers:
14         if num % 2 == 0:
15             even_sum += num
16         else:
17             odd_sum += num
18
19     return odd_sum, even_sum
20
21
22 if __name__ == "__main__":
23     # Test the function
24     numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
25
26     odd_sum, even_sum = calculate_sums(numbers)
27
28     print(f"List: {numbers}")
29     print(f"Sum of odd numbers: {odd_sum}")
30     print(f"Sum of even numbers: {even_sum}")
31     print(f"Total sum: {odd_sum + even_sum}")
32
```

The screenshot shows a terminal window with a dark theme. The command 'python3.11.exe' is run in the directory 'C:/AI coding/Assignment 1.2/Assignment 2.5'. The script is executed, and its output is displayed. The output shows the list of numbers [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], and the sums of odd and even numbers, which both total 55.

```
PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI coding/Assignment 1.2/Assignment 2.5/main.py"
y"
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y"
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of odd numbers: 25
Sum of even numbers: 30
Sum of even numbers: 30
Total sum: 55
PS C:\AI coding>
```

CHAT

+ ⌂ ⚙ ... | ⌂ X

← SUM OF ODD AND EVEN NUMBERS PRO... ⌂

Refactor this code to make it more efficient and readable.

✓ Read main.py file

I see the file has reverted to the original version. Let me refactor it again with improvements:

✓ Edited main.py +10 -14

Done! I've refactored the code with the following improvements:

Enhancements:

- **More Pythonic:** Uses generator expressions with `sum()` instead of manual loops
- **Better organization:** Separated calculation logic from display logic into two functions
- **More reusable:** The `display_results()` function encapsulates output formatting
- **Cleaner main block:** Simplified to a single function call



+ main.py

Describe what to build next

Agent ⌂ Auto ⌂



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files: main.py (selected), factorial_with_function.py, and factorial.py. The main editor area displays Python code for calculating the sum of odd and even numbers in a list. The terminal at the bottom shows the execution of the script and its output.

```
Assignment 1.2 > Assignment 2.5 > main.py > ...
1 def calculate_sums(numbers):
2     """Calculate the sum of odd and even numbers in a list.
3
4     Args:
5         numbers: A list of integers
6
7     Returns:
8         A tuple of (odd_sum, even_sum)
9     """
10    odd_sum = sum(num for num in numbers if num % 2)
11    even_sum = sum(num for num in numbers if num % 2 == 0)
12    return odd_sum, even_sum
13
14
15 def display_results(numbers):
16     """Prints a list of numbers and their sums for odd and even numbers.
17     Args:
18         numbers: A list of integers
19
20     Returns:
21         A tuple of (odd_sum, even_sum)
22     """
23     odd_sum, even_sum = calculate_sums(numbers)
24     total_sum = odd_sum + even_sum
25
26     print(f"List: {numbers}")
27     print(f"Sum of odd numbers: {odd_sum}")
28     print(f"Sum of even numbers: {even_sum}")
29     print(f"Total sum: {total_sum}")
30
31
32 if __name__ == "__main__":
33     display_results([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
34
```

```
PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI coding/Assignment 1.2/Assignment 2.5/main.py"
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30
Total sum: 55
PS C:\AI coding>
```

EXPLANATION:

The refactored code uses Python's built-in `sum()` function with conditions.

This reduces unnecessary variables and improves readability.

The logic is clearer and more efficient than the original version.

TASK_02:

Scenario

You are onboarding a junior developer.

Task Description

Ask Gemini to explain a function that calculates the area of different shapes.

The screenshot shows a code editor interface with a dark theme. At the top, there is a task card with the title "Explain a Python function that calculates the area of different shapes." Below the task card, the code editor displays the following text:

✓ Created program to calculate areas of shapes
I'll create a Python program that calculates the area of different shapes and explain how it works:
✓ Created `shapes.py`
I've created a program with shape area calculation functions. Here's an explanation:

Key Functions:

1. `circle_area(radius)` - Uses the formula: $\pi \times r^2$
 - o Takes radius as input, returns the area using `math.pi`
2. `rectangle_area(length, width)` - Uses the formula: $length \times width$
 - o Simple multiplication of two dimensions
3. `triangle_area(base, height)` - Uses the formula: $\frac{1}{2} \times base \times height$
 - o The classic triangle area formula with 0.5 multiplier
4. `square_area(side)` - Uses the formula: $side^2$
 - o Calculates area by squaring one side length
5. `ellipse_area(major_axis, minor_axis)` - Uses the formula: $\pi \times a \times b$
 - o Where a and b are the semi-major and semi-minor axes

At the bottom of the code editor, there are icons for file operations (New, Open, Save) and a status bar that says "Describe what to build next".

The screenshot shows a code editor interface with a dark theme. The file `shapes.py` is open, displaying the following Python code:

```
1 import math
2 def circle_area(radius):
3     return math.pi * radius ** 2
4 def rectangle_area(length, width):
5     return length * width
6 def triangle_area(base, height):
7     return 0.5 * base * height
8 def square_area(side):
9     return side ** 2
10 def ellipse_area(major_axis, minor_axis):
11     return math.pi * major_axis * minor_axis
```

```

11     if __name__ == "__main__":
12         print("=" * 50)
13         print("AREA CALCULATOR FOR DIFFERENT SHAPES")
14         print("=" * 50)
15         # Circle
16         radius = 5
17         area = circle_area(radius)
18         print(f"\nCircle with radius {radius}: Area = {area:.2f}")
19         # Rectangle
20         length, width = 10, 6
21         area = rectangle_area(length, width)
22         print(f"Rectangle ({length} x {width}): Area = {area:.2f}")
23         # Triangle
24         base, height = 8, 5
25         area = triangle_area(base, height)
26         print(f"Triangle (base={base}, height={height}): Area = {area:.2f}")
27         # Square
28         side = 7
29         area = square_area(side)
30         print(f"Square with side {side}: Area = {area:.2f}")
31         # Ellipse
32         major, minor = 10, 5
33         area = ellipse_area(major, minor)
34         print(f"\nEllipse (major={major}, minor={minor}): Area = {area:.2f}")
35         print("\n" + "=" * 50)
36

```

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
+  ...  |  X
● PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI coding/Assignment 1.2/Assignment 2.5/shapes.py"
=====
AREA CALCULATOR FOR DIFFERENT SHAPES
=====

Circle with radius 5: Area = 78.54
Rectangle (10 x 6): Area = 60.00
Triangle (base=8, height=5): Area = 20.00
Square with side 7: Area = 49.00
Ellipse (major=10, minor=5): Area = 157.08

Ellipse (major=10, minor=5): Area = 157.08
=====
PS C:\AI coding>

```

AI Explanation

This function calculates the area of different shapes.

- For a circle, it uses the formula $\pi \times r^2$
- For a rectangle, it uses $length \times width$
- For a triangle, it uses $\frac{1}{2} \times base \times height$...The function selects the correct formula using conditional statements

TASK_03:

Scenario

You are testing how AI responds to different prompts.

Task Description

Use Cursor AI with different prompts for the same problem and observe code changes.

```
import math

def is_prime_simple(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

def is_prime_optimized(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

if __name__ == "__main__":
    test_numbers = [0, 1, 2, 3, 4, 16, 17, 19, 20, 97, 7919, 7920]

    print("Number | Simple | Optimized")
    print("-----")
    for num in test_numbers:
        simple = is_prime_simple(num)
        opt = is_prime_optimized(num)
        print(f"{num:6} | {str(simple):6} | {str(opt):9}")
    large = 104729 # 10000th prime
    print("\nChecking a larger number:")

Ln 15, Col 21 Spaces: 4 UTF-8
```

```
PS C:\AI coding>

...
PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI coding/A
ssignment 1.2/Assigment 2.5/prime_check.py"
PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI coding/A
ssignment 1.2/Assigment 2.5/prime_check.py"
● Number | Simple | Optimized
-----
  0 | False | False
  1 | False | False
  2 | True | True
  3 | True | True
  4 | False | False
  16 | False | False
  17 | True | True
  19 | True | True
  20 | False | False
  97 | True | True
  7919 | True | True
  7920 | False | False

Checking a larger number:
104729 is prime (optimized): True
○ PS C:\AI coding>
```

Observation

Different prompts produce different styles of code:

- Simple
- Optimized
- Commented

Prompt wording strongly affects AI output quality.

TASK_04:

Scenario

You must recommend an AI coding tool.

Task Description

Compare Gemini, Copilot, and Cursor AI for usability and code quality.

GitHub Copilot, Google Gemini, and Cursor AI are all useful AI coding tools, but each has different strengths.

GitHub Copilot works directly inside VS Code and provides real-time code suggestions while typing. It is best for fast development and daily programming tasks.

Google Gemini in Google Colab is more useful for explanations and understanding concepts. It helps beginners learn coding logic clearly.

Cursor AI is good for testing different prompts, refactoring code, and improving code quality. It allows users to experiment with multiple coding styles.

conclusion:

- **Copilot** is best for speed
- **Gemini** is best for learning
- **Cursor AI** is best for experimentation