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
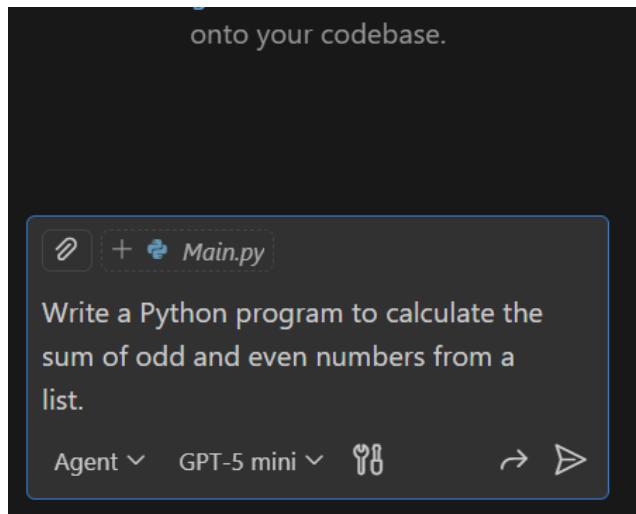
Name of Student : R.Sowmya Sri

Enrollment No. : 2303A52105

Batch No. : 33

Date :09/01/26

TASK-1

+ 🐍 *Main.py*

Write a Python program to calculate the sum of odd and even numbers from a list.

Agent ∨   GPT-5 mini ∨   🛠
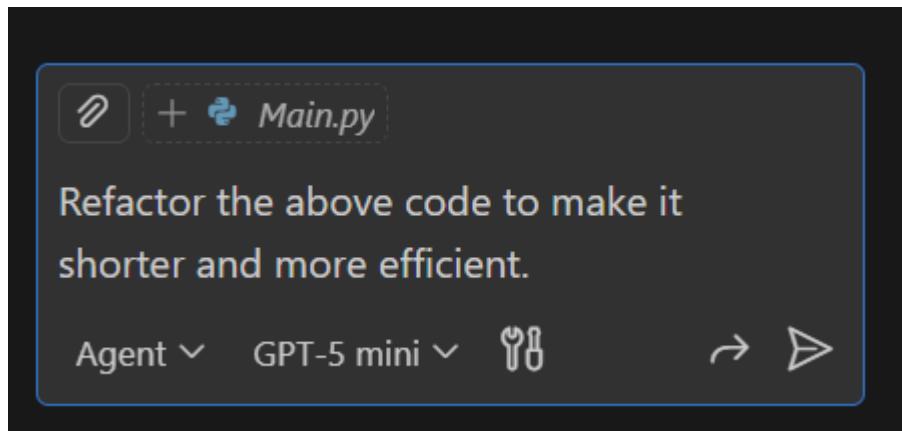
```python
def sum_even_odd(numbers):
    """Return a tuple (sum_even, sum_odd) for the iterable of integers."""
    sum_even = 0
    sum_odd = 0
    for n in numbers:
        try:
            num = int(n)
        except (TypeError, ValueError):
            continue
        if num % 2 == 0:
            sum_even += num
        else:
            sum_odd += num
    return sum_even, sum_odd


if __name__ == "__main__":
    sample = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    print("Sample list:", sample)
    even_sum, odd_sum = sum_even_odd(sample)
    print(f"Sum of even numbers: {even_sum}")
    print(f"Sum of odd numbers: {odd_sum}")

    # optional: let user provide numbers
    s = input("Enter integers separated by space (or press Enter to skip): ")
    if s.strip():
        parts = s.split()
        even_sum, odd_sum = sum_even_odd(parts)
        print("For your input:")
        print(f"Sum of even numbers: {even_sum}")
        print(f"Sum of odd numbers: {odd_sum}")
```

```
Sum of odd numbers: 25
PS S:\3 2 sem\AIAssisted_2026> & C:/Users/sowmy/AppData/Local/Microsoft/WindowsApps/python3.11.exe "s:/
3 2 sem/AIAssisted_2026/Assignment2.5/Main.py"
Sample list: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Sum of even numbers: 20
Sum of odd numbers: 25
Enter integers separated by space (or press Enter to skip): 22 34 56 77 66
For your input:
Sum of even numbers: 178
Sum of odd numbers: 77
PS S:\3 2 sem\AIAssisted_2026>
```

Python
Python

Refactoring Prompt


```
def sum_even_odd(numbers):
    Return (sum_even, sum_odd) for an iterable of ints or int-like strings.

    Non-convertible items are ignored. Single-pass, memory-efficient.
    """
    even = odd = 0
    for x in numbers:
        try:
            n = int(x)
        except (TypeError, ValueError):
            continue
        if n & 1:
            odd += n
        else:
            even += n
    return even, odd


if __name__ == "__main__":
    sample = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    print("Sample list:", sample)
    even_sum, odd_sum = sum_even_odd(sample)
    print(f"Sum of even numbers: {even_sum}")
    print(f"Sum of odd numbers: {odd_sum}")

    s = input("Enter integers separated by space (or press Enter to skip): ")
    if s.strip():
        even_sum, odd_sum = sum_even_odd(s.split())
        print("For your input:")
        print(f"Sum of even numbers: {even_sum}")
        print(f"Sum of odd numbers: {odd_sum}")
```

```
Sample list: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Sum of even numbers: 20
Sum of odd numbers: 25
Enter integers separated by space (or press Enter to skip): 1 3 4 5 6 7 8 9 10
For your input:
Sum of even numbers: 28
Sum of odd numbers: 25
PS S:\3 2 sem\AIAssisted_2026>
```

AI reduced the code size, removed loops and improved readability.

TASK-1

Write a function to calculate the area of a circle, rectangle and triangle and explain the code

```python
from math import pi
def area_circle(radius):
    """Return area of a circle given its radius.

    Uses formula: area = pi * r^2
    """
    return pi * (radius ** 2)


def area_rectangle(width, height):
    """Return area of a rectangle (width * height)."""
    return width * height


def area_triangle(base, height):
    """Return area of a triangle using (base * height) / 2."""
    return 0.5 * base * height


def _demo():
    c = area_circle(3)
    r = area_rectangle(4, 5)
    t = area_triangle(6, 7)
    print(f"Circle area (r=3): {c}")
    print(f"Rectangle area (4x5): {r}")
    print(f"Triangle area (base=6,height=7): {t}")


if __name__ == "__main__":
    _demo()
```
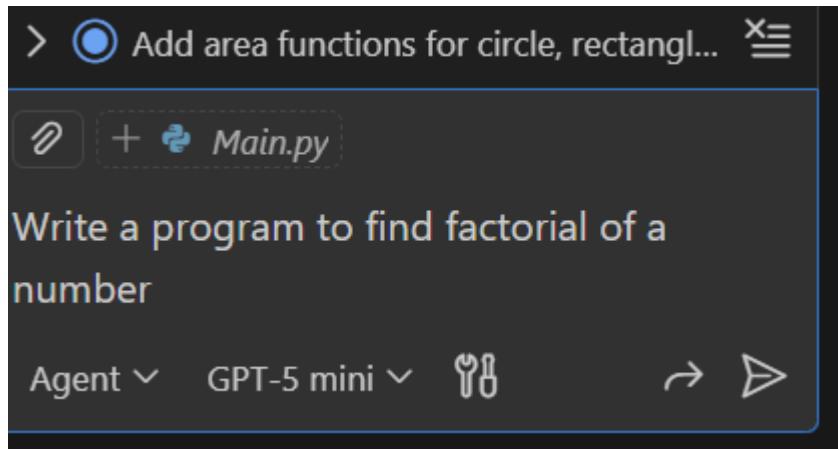
TASK-3

Prompt 1


```
Assignment2.5 >   Main.py > ...
1    def factorial(n):
2        """Return n! for a non-negative integer n.
3
4        Accepts integer-like inputs (e.g. 5, 5.0, '5'). Raises ValueError
5        for negative or non-integer inputs.
6        """
7        if not float(n).is_integer():
8            raise ValueError("Factorial requires an integer input")
9        n = int(n)
10       if n < 0:
11           raise ValueError("Factorial is not defined for negative numbers")
12       result = 1
13       for i in range(2, n + 1):
14           result *= i
15       return result
16
17
18   if __name__ == "__main__":
19       try:
20           s = input("Enter a non-negative integer to compute factorial (or press Enter to show exa
21           if not s.strip():
22               print("5! =", factorial(5))
23           else:
24               val = int(s.strip())
25               print(f"{val}! =", factorial(val))
26       except ValueError as e:
27           print("Invalid input:", e)
28   |
```

```
Enter a non-negative integer to compute factorial (or press Enter to show example 5!): 7
7! = 5040
PS S:\3 2 sem\AIAssisted_2026> |
```

Prompt 2

```python
import math
def factorial(n):
    """Return n! for a non-negative integer n.

    Optimization: delegates computation to `math.factorial` (C-optimized).

    Validation/acceptance rules:
    - Accepts `int`, `float` that is an integer (e.g. 5.0), or numeric `str` like '5' or '5.0'.
    - Raises `ValueError` for non-integer, negative, or otherwise invalid inputs.
    """
    # Try to coerce common input types to an integer, with validation
    try:
        if isinstance(n, str):
            s = n.strip()
            if s == "":
                raise ValueError
            # allow '5', '5.0', ' 5 ' and scientific notation like '1e2'
            f = float(s)
            if not f.is_integer():
                raise ValueError("Factorial requires an integer input")
            n_int = int(f)
        elif isinstance(n, float):
            if not n.is_integer():
                raise ValueError("Factorial requires an integer input")
            n_int = int(n)
        elif isinstance(n, int):
```

```python
        if not n.is_integer():
            raise ValueError("Factorial requires an integer input")
        n_int = int(n)
    elif isinstance(n, int):
        n_int = n
    else:
        # try a best-effort conversion for numeric-like objects
        n_int = int(n)
        if float(n) != float(n_int):
            raise ValueError("Factorial requires an integer input")
except (ValueError, TypeError):
    raise ValueError("Factorial requires an integer-like non-negative input")

if n_int < 0:
    raise ValueError("Factorial is not defined for negative numbers")

# Use math.factorial for best performance and big-integer support
return math.factorial(n_int)


if __name__ == "__main__":
    try:
        s = input("Enter a non-negative integer to compute factorial (or press Enter to show exa
        if not s.strip():
            print("5! =", factorial(5))
        else:
            print(f"{s.strip()}! =", factorial(s.strip()))
    except ValueError as e:
        print("Invalid input:", e)
```

```
Enter a non-negative integer to compute factorial (or press Enter to show example 5!): 8
8! = 40320
PS S:\3 2 sem\AIAssisted_2026>
```

Observation

Different prompts produced different code structures, logic and features.

<u>Task 4:</u> Tool Comparison Reflection

Scenario:

You must recommend an AI coding tool.

Task:

Based on the work done in this lab, Gemini, Copilot, and Cursor AI were compared in terms of usability and code quality.

Short Written Reflection:

After using all three AI tools, it was observed that each tool serves a different purpose.
Google Gemini is very useful for learning and understanding concepts because it provides detailed explanations and well-structured code. It is best suited for beginners and documentation work.

GitHub Copilot is excellent for fast coding. It suggests code directly inside the IDE, which saves time and improves productivity. However, it does not provide much explanation, so it is better suited for experienced programmers.

Cursor AI performs very well in code refactoring and optimization. It allows users to experiment with different prompts and observe changes in the code, making it useful for improving code quality and testing different approaches.

Final Recommendation:

For learning and concept clarity — **Gemini**
For fast development — **Copilot**
For code improvement and experimentation — **Cursor AI**