AI ASSISTANT CODING
LAB-02

Name : P. Sricharan Goud

Roll No : 2303A52100

Batch No : 33

Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Google Gemini (Colab) and Cursor AI

Task – 1 : Refactoring Odd/Even Logic (List Version)

Scenario: You are improving legacy code.

The given legacy program calculates the sum of odd and even numbers in a list. The code needs to be improved for better readability and efficiency using AI tools.

Prompt Used :

Write a program to calculate the sum of odd and even numbers in a list,
Refactor this Python code to improve readability and efficiency.

Original Code :

```
def sum_odd_even(numbers: list[int]) -> tuple[int, int]:
    even_sum = 0
    odd_sum = 0
    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    return even_sum, odd_sum
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum, odd_sum = sum_odd_even(my_list)
print(f"List: {my_list}")
print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

AI-Refactored Code :

```
from typing import Iterable, Tuple


def sum_even_odd(numbers: Iterable[int]) -> Tuple[int, int]:
    """Return a tuple (even_sum, odd_sum) for the given iterable of integers.


    This performs a single-pass accumulation for best efficiency.
    """
    even_sum = 0
    odd_sum = 0
    for n in numbers:
        if n & 1:  # odd when least-significant bit is 1
            odd_sum += n
        else:
            even_sum += n
    return even_sum, odd_sum


def main() -> None:
    sample = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    even, odd = sum_even_odd(sample)
    print(f"Input: {sample}")
    print(f"Sum of even numbers: {even}")
    print(f"Sum of odd numbers: {odd}")
if __name__ == "__main__":
    main()
```

REFACTORING PYTHON CODE ...

Write a program to calculate the sum of odd and even numbers in a list,
Refactor this Python code to improve readability and efficiency.

✓ Optimizing tool selection...

I'll create a short TODO plan for the refactor, then inspect 2.5.py.

✓ Managed and tracked todo items for planning

Now I'll update 2.5.py with a clean, single-pass implementation and example usage.

✓ Edited 2.5.py +30 -0

✓ Configuring Python environment

I'll configure the Python environment so I can run the script.

✓ Configured Python environment

✓ Creating a Virtual Environment

✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

Todos (5/5)

2.5.py +

Describe what to build next

Agent Auto

2.5.py > ...

```python

from typing import Iterable, Tuple


def sum_even_odd(numbers: Iterable[int]) -> Tuple[int, int]:
    """Return a tuple (even_sum, odd_sum) for the given iterable of integers.

    This performs a single-pass accumulation for best efficiency.
    """
    even_sum = 0
    odd_sum = 0
    for n in numbers:
        if n & 1:  # odd when least-significant bit is 1
            odd_sum += n
        else:
            even_sum += n
    return even_sum, odd_sum


def main() -> None:
    sample = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    even, odd = sum_even_odd(sample)
    print(f"Input: {sample}")
    print(f"Sum of even numbers: {even}")
    print(f"Sum of odd numbers: {odd}")


if __name__ == "__main__":
    main()

```

```
PS C:\Users\Pedıripati Sricharan\AIAC> & "C:\Users\Pedıripati Sricharan\AIAC\.venv\Scripts\python.exe" "c:/Users/Pedıripati Sricharan/AIAC/2.5.py"
circle (r=3): 28.274333882308138
rectangle (4x5): 20.0
triangle (base=6, height=2): 6.0
PS C:\Users\Pedıripati Sricharan\AIAC> 
```

Explanation :

The refactored code reduces the number of variables and uses Python’s built-in sum() function. It improves readability, reduces code length, and enhances maintainability. The logic is clearer and easier to modify.

Task 2: Area Calculation Explanation (Gemini)

Scenario

This task helps onboard a junior developer by explaining a function that calculates areas of different shapes.

Prompt Used :

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

```
import math


def calculate_area(shape: str, **kwargs) -> float:
    """
    Calculates the area of different geometric shapes.


    Args:
        shape (str): The type of shape ('circle', 'rectangle', 'triangle').
        **kwargs: Keyword arguments for the dimensions of the shape:
            - For 'circle': 'radius' (float)
            - For 'rectangle': 'length' (float), 'width' (float)
            - For 'triangle': 'base' (float), 'height' (float)


    Returns:
        float: The calculated area of the shape.


    Raises:
```

```python
        ValueError: If an unsupported shape is provided or required dimensions are missing.
    """
    if shape.lower() == 'circle':
        if 'radius' in kwargs:
            radius = kwargs['radius']
            return math.pi * radius**2
        else:
            raise ValueError("For 'circle', 'radius' is required.")
    elif shape.lower() == 'rectangle':
        if 'length' in kwargs and 'width' in kwargs:
            length = kwargs['length']
            width = kwargs['width']
            return length * width
        else:
            raise ValueError("For 'rectangle', 'length' and 'width' are required.")
    elif shape.lower() == 'triangle':
        if 'base' in kwargs and 'height' in kwargs:
            base = kwargs['base']
            height = kwargs['height']
            return 0.5 * base * height
        else:
            raise ValueError("For 'triangle', 'base' and 'height' are required.")
    else:
        raise ValueError("Unsupported shape. Choose from 'circle', 'rectangle', or 'triangle'.")

# Example usage:
print(f"Area of circle (radius=5): {calculate_area('circle', radius=5):.2f}")
print(f"Area of rectangle (length=4, width=6): {calculate_area('rectangle', length=4, width=6):.2f}")
print(f"Area of triangle (base=7, height=3): {calculate_area('triangle', base=7, height=3):.2f}")

# Example of handling an error (uncomment to test):
# try:
#     print(calculate_area('square', side=5))
# except ValueError as e:
#     print(f"Error: {e}")
```

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

I'll create a short TODO plan, check the current 2.5.py, then add the area functions and run a quick example.

✓ Managed and tracked todo items

✓ Edited 2.5.py +67 -1

✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

✓ Starting: *Mark tasks completed and explain code* (5/5)

Progress: added area functions in 2.5.py:1-200 and verified example output.

Explanation — concise:

- `area_circle(radius)`: computes A = π * r^2. Uses `_validate_non_negative` to coerce to float and reject negative radii.

> Mark tasks completed and...

2.5.py +

Describe what to build next

Agent ⌄ Auto ⌄

2.5.py > ...

```python
"""Geometry helpers: area calculations for circle, rectangle, and triangle.

Functions:
- area_circle(radius)
- area_rectangle(width, height)
- area_triangle(base, height)

Each function validates inputs (non-negative) and returns a float.
"""

from __future__ import annotations
from math import pi
from typing import Union

Number = Union[int, float]


def _validate_non_negative(name: str, value: Number) -> float:
    v = float(value)
    if v < 0:
        raise ValueError(f"{name} must be non-negative, got {value}")
    return v


def area_circle(radius: Number) -> float:
    """Return area of a circle: A = π * radius^2.

    Raises ValueError for negative radius.
    """
    r = _validate_non_negative("radius", radius)
    return pi * r * r


def area_rectangle(width: Number, height: Number) -> float:
    """Return area of a rectangle: A = width * height.

    Raises ValueError for negative dimensions.
    """
```

```python
    w = _validate_non_negative("width", width)
    h = _validate_non_negative("height", height)
    return w * h


def area_triangle(base: Number, height: Number) -> float:
    """Return area of a triangle: A = 0.5 * base * height.

    Raises ValueError for negative inputs.
    """
    b = _validate_non_negative("base", base)
    h = _validate_non_negative("height", height)
    return 0.5 * b * h


def _example() -> None:
    examples = (
        ("circle r=3", area_circle(3)),
        ("rectangle 4x5", area_rectangle(4, 5)),
        ("triangle base=6 height=2", area_triangle(6, 2)),
    )
    for name, area in examples:
        print(f"{name}: {area}")


if __name__ == "__main__":
    _example()
```

```
PS C:\Users\Pediripati Sricharan\AIAC> & "C:\Users\Pediripati Sricharan\AIAC\.venv\Scripts\python.exe" "c:/Users/Pediripati Sricharan/AIAC/2.5.py"
circle r=3: 28.274333882308138
rectangle 4x5: 20.0
triangle base=6 height=2: 6.0
PS C:\Users\Pediripati Sricharan\AIAC>
```

Explanation :

The function accepts the shape name and required dimensions as input. Conditional statements decide which formula to apply. For a circle, radius is used, while rectangle and triangle require two values. The function returns the calculated area based on the selected shape.

Task 3: Prompt Sensitivity Experiment (Cursor AI)

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

The factorial problem was used to test how different prompts affect AI-generated code.

Prompts Used :

1. Write a Python program to calculate factorial
2. Write an optimized Python function to calculate factorial with comments
3. Generate a beginner-friendly factorial program with error handling

1. Write a Python program to calculate factorial
2. Write an optimized Python function to calculate factorial with comments
3. Generate a beginner-friendly factorial program with error handling

✓ Inspected file changes and implemented factorial functions

I'll add three factorial implementations to `2.5.py` : simple, optimized, and beginner-friendly with error handling — then run examples.

✓ Managed and tracked todo items

Now I'll overwrite `2.5.py` with a complete file containing area functions and the three factorial implementations, then run examples.

Inspect `2.5.py` (1/5)

2.5.py +

Agent Auto

C:\Users\Pediripati Sricharan\AIAC\2.5.py

```python
"""Utilities: area calculations and factorial examples.

Contains:
- area_circle(radius), area_rectangle(width, height), area_triangle(base, height)
- factorial_simple(n): straightforward iterative factorial
- factorial_optimized(n): uses math.prod for speed
- factorial_beginner(): CLI-friendly function with input validation
"""

from __future__ import annotations
from math import pi
from typing import Union

Number = Union[int, float]


def _validate_non_negative(name: str, value: Number) -> float:
    v = float(value)
    if v < 0:
        raise ValueError(f"{name} must be non-negative, got {value}")
    return v


def area_circle(radius: Number) -> float:
    """Return area of a circle: A = π * radius^2.

    Raises ValueError for negative radius.
    """
    r = _validate_non_negative("radius", radius)
    return pi * r * r


def area_rectangle(width: Number, height: Number) -> float:
    """Return area of a rectangle: A = width * height.

    Raises ValueError for negative dimensions.
    """
    w = _validate_non_negative("width", width)
    h = _validate_non_negative("height", height)
```

```python
def area_triangle(base: Number, height: Number) -> float:
    """Return area of a triangle: A = 0.5 * base * height.

    Raises ValueError for negative inputs.
    """
    b = _validate_non_negative("base", base)
    h = _validate_non_negative("height", height)
    return 0.5 * b * h


# --------------- Factorial implementations ---------------


def factorial_simple(n: int) -> int:
    """Simple factorial calculation using an iterative loop.

    Returns n! for integer n >= 0. Raises ValueError for negative input.
    Clear and easy to read; good for teaching and small-to-medium n.
    """
    if not isinstance(n, int):
        raise TypeError("n must be an integer")
    if n < 0:
        raise ValueError("n must be non-negative")
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

```

```python
def factorial_optimized(n: int) -> int:
    """Optimized factorial using `math.prod` for performance.

    Rationale:
    - `math.prod` performs the product at C-speed over the iterable, often
      faster than a Python-level loop for large n.
    - Still validates inputs and returns exact integer results.
    """
    from math import prod

    if not isinstance(n, int):
        raise TypeError("n must be an integer")
    if n < 0:
        raise ValueError("n must be non-negative")
    if n < 2:
        return 1
    return prod(range(1, n + 1))


def factorial_beginner() -> None:
    """Beginner-friendly CLI: reads input, handles errors, prints factorial.

    This function is interactive and intended for demonstration. It accepts
    an input string, validates it, and prints a helpful message on errors.
    """
    try:
        s = input("Enter a non-negative integer to compute factorial: ").strip()
        if not s:
            print("No input provided. Exiting.")
            return
        n = int(s)
    except ValueError:
        print("Invalid input: please enter a whole non-negative integer.")
        return
```

```python
    try:
        result = factorial_optimized(n)
    except ValueError as ve:
        print(f"Error: {ve}")
        return

    print(f"{n}! = {result}")


def _example_areas() -> None:
    examples = (
        ("circle r=3", area_circle(3)),
        ("rectangle 4x5", area_rectangle(4, 5)),
        ("triangle base=6 height=2", area_triangle(6, 2)),
    )
    for name, area in examples:
        print(f"{name}: {area}")


def _example_factorials() -> None:
    examples = (0, 1, 5, 10)
    for n in examples:
        print(f"{n}! (simple) = {factorial_simple(n)}")
        print(f"{n}! (optimized) = {factorial_optimized(n)}")


if __name__ == "__main__":
    _example_areas()
    print("---")
    _example_factorials()

```

```
PS C:\Users\Pediripati Sricharan\AIAC> & "C:\Users\Pediripati Sricharan\AIAC\.venv\Scripts\python.exe" "c:/Users/Pediripati Sricharan/AIAC/2.5.py"
circle r=3: 28.274333882308138
rectangle 4x5: 20.0
triangle base=6 height=2: 6.0
---
0! (simple) = 1
0! (optimized) = 1
1! (simple) = 1
1! (optimized) = 1
5! (simple) = 120
5! (optimized) = 120
10! (simple) = 3628800
10! (optimized) = 3628800
PS C:\Users\Pediripati Sricharan\AIAC>
```

Observation

The AI produced different code styles depending on the prompt. Optimized prompts improved performance and readability, while beginner-friendly prompts added validation and safety checks.

Task 4: Tool Comparison Reflection

Comparison of Gemini, Copilot, and Cursor AI

Reflection :

Google Gemini is best suited for explanations and learning support. GitHub Copilot provides real-time inline suggestions, improving developer productivity. Cursor AI excels in experimentation, refactoring, and prompt-based exploration. Each tool serves a different purpose, and choosing the right one depends on whether the goal is learning, development speed, or code analysis