ASSIGNMENT-2.5

M.HARIKRISHNA

2203A52100

B-50

Task 1

Refactoring Odd/Even Logic (List Version)

❖ Scenario:

You are improving legacy code.

❖ Task:

Write a program to calculate the sum of odd and even numbers in a list,

then refactor it using AI.

❖ Expected Output: ❖ Original and improved code

Prompt:

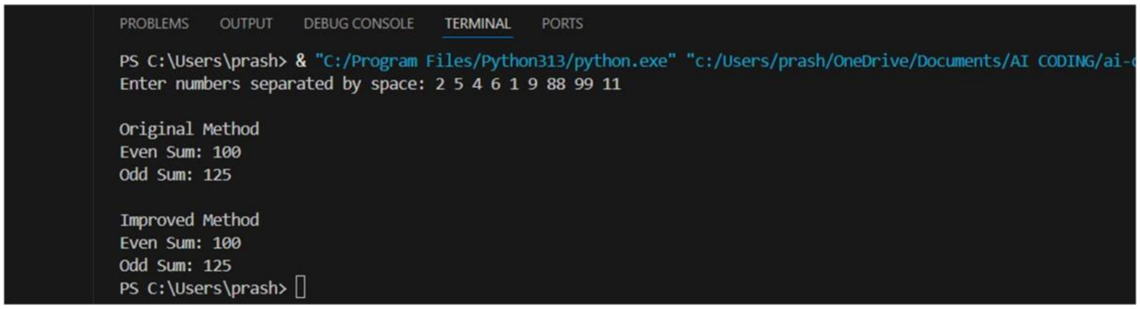
Generate a Python program to calculate the sum of odd and even numbers in a list using loops, then refactor it to improve readability, efficiency, and Pythonic style..

Code:

```
# Task 1: Odd/Even Sum with User Input
# Take list input from user numbers = list(map(int, input("Enter numbers
separated by space: ").split()))
# --- Original Logic ---
even_sum = 0
odd_sum = 0 for num
in numbers:     if num
% 2 == 0:
even_sum += num
   else:
       odd_sum += num
```

```
print("\nOriginal Method")
print("Even Sum:", even_sum)
print("Odd Sum:", odd_sum)


# --- Improved Pythonic Logic --- even_sum_new = sum(num
for num in numbers if num % 2 == 0) odd_sum_new =
sum(num for num in numbers if num % 2 != 0)
print("\nImproved Method") print("Even Sum:",
even_sum_new) print("Odd Sum:", odd_sum_new)
```

Output:



```
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDrive/Documents/AI CODING/ai-
Enter numbers separated by space: 2 5 4 6 1 9 88 99 11

Original Method
Even Sum: 100
Odd Sum: 125

Improved Method
Even Sum: 100
Odd Sum: 125
PS C:\Users\prash> 
```

Code Explanation:

In this task, the original code uses a loop and conditional statements to check whether each number is odd or even and adds it to the respective sum. The improved version uses Python’s built-in sum() function and generator expressions to make the code shorter, more readable, and efficient. The optimized version reduces unnecessary variables and improves performance, especially when working with large lists

---

Task 2

Task 2: Area Calculation Explanation

❖ Scenario:

You are onboarding a junior developer.

❖ Task:

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

❖ Expected Output:

➢ Code ➢

Explanation

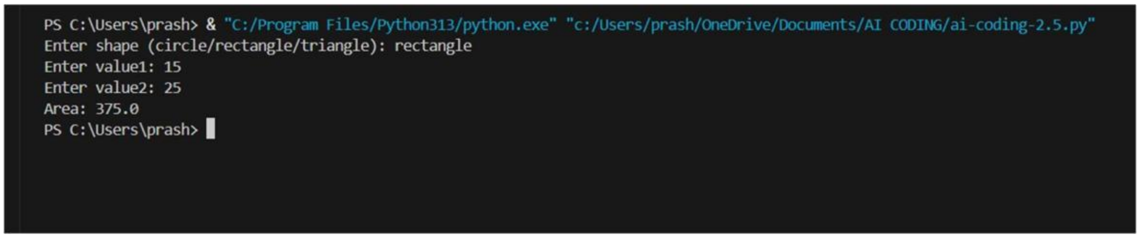
Prompt:

Generate a Python function to calculate the area of different shapes like circle, rectangle, and triangle, and provide explanation of the logic.

Code:

```
# Task 2: Area Calculation with User Input

import math def calculate_area(shape,

value1, value2=0):

    if shape == "circle":

        return math.pi * value1 *

value1      elif shape == "rectangle":

return value1 * value2      elif shape

== "triangle":

        return 0.5 * value1 * value2

else:

        return "Invalid Shape" shape = input("Enter shape

(circle/rectangle/triangle): ").lower() if shape == "circle":

    r = float(input("Enter radius: "))

print("Area:", calculate_area(shape, r)) elif

shape in ["rectangle", "triangle"]:      v1 =

float(input("Enter value1: "))      v2 =

float(input("Enter value2: "))

print("Area:", calculate_area(shape, v1, v2))

else:

    print("Invalid Shape")
```

Output:



```
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDrive/Documents/AI CODING/ai-coding-2.5.py"
Enter shape (circle/rectangle/triangle): rectangle
Enter value1: 15
Enter value2: 25
Area: 375.0
PS C:\Users\prash> 
```

Code Explanation:

This function calculates the area based on the shape type. It uses conditional statements to identify which formula to apply. For circle, it uses πr². For rectangle, it multiplies length and width. For triangle, it uses ½ × base × height. This design makes the function reusable and easy to extend if new shapes need to be added.

Task 3

Task 3: Prompt Sensitivity Experiment

❖ Scenario:

You are testing how AI responds to different prompts.

❖ Task:

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

code changes.

❖ Expected Output:

➢ Prompt list

➢ Code variations

Prompt:

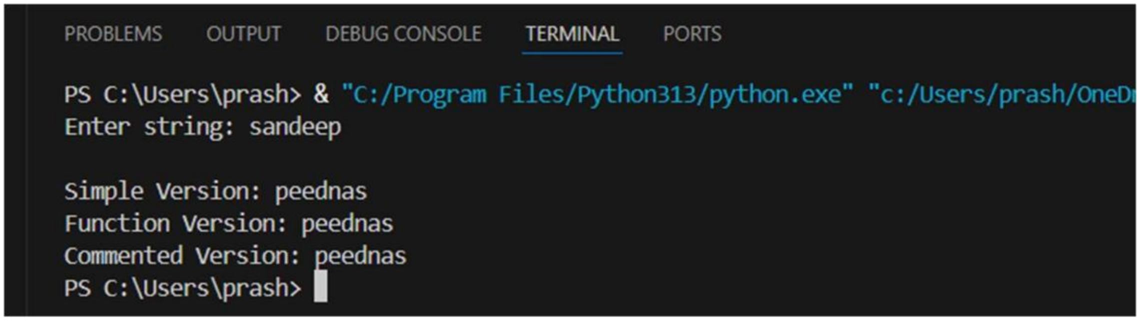
Generate Python code to reverse a string using different prompt styles such as simple prompt, optimized prompt, and function-based prompt, and compare the outputs.

Code:

```
# Task 3: String Reversal Variations with User Input

text = input("Enter string: ")


# Simple Version
```

```
print("\nSimple Version:", text[::-1])


# Function Version
def reverse_string(s):
   return s[::-1] print("Function Version:",
reverse_string(text))


# Commented Version def reverse_string_commented(text):
# Reverse string using slicing          return text[::-1]
print("Commented                                    Version:",
reverse_string_commented(text))
```

Output:



```
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDr
Enter string: sandeep

Simple Version: peednas
Function Version: peednas
Commented Version: peednas
PS C:\Users\prash>
```

Code Explanation:

Different prompts produce different types of code. A simple prompt usually generates basic code. An optimized prompt generates efficient and cleaner code. A function-based prompt generates reusable and structured code. This experiment shows how prompt wording affects AI-generated code quality and structure.

Task 4

T ask 4: Tool Comparison Reflection ❖

Scenario:

You must recommend an AI coding tool.

❖ Task:

Based on your work in this topic, compare Gemini, Copilot, and Cursor AI for usability and code quality.

❖ Expected Output: Short written reflection

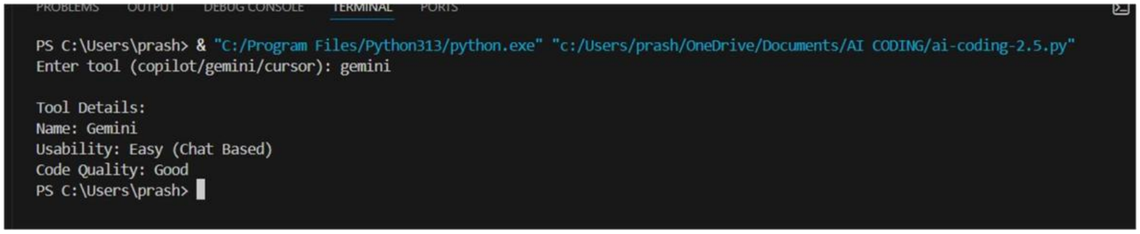
Prompt:

Compare Gemini, GitHub Copilot, and Cursor AI based on usability, code quality, and learning support.

Code:

```
# Task 4: AI Tool Comparison with User Input


tools = {
    "copilot": {
        "Name": "GitHub Copilot",
        "Usability": "Very Easy (IDE Integrated)",
        "Code Quality": "High"
    },
    "gemini": {
        "Name": "Gemini",
        "Usability": "Easy (Chat Based)",
        "Code Quality": "Good"
    },
    "cursor": {
        "Name": "Cursor AI",
        "Usability": "Moderate",
        "Code Quality": "Good"
    }
}


choice = input("Enter tool (copilot/gemini/cursor): ").lower()


if choice in tools:
    print("\nTool Details:")     for key,
value in tools[choice].items():
        print(f"{key}: {value}")
else:
    print("Invalid Tool Choice")
```

Output:



```
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDrive/Documents/AI CODING/ai-coding-2.5.py"
Enter tool (copilot/gemini/cursor): gemini

Tool Details:
Name: Gemini
Usability: Easy (Chat Based)
Code Quality: Good
PS C:\Users\prash> 
```

Code explanation:

Gemini is strong in explanations and concept learning. GitHub Copilot is best for real-time coding assistance inside IDEs and provides high-quality code suggestions. Cursor AI is good for prompt-based complete code generation and editing. Each tool is useful depending on the user’s need, such as learning, coding speed, or full project generation.