AI Assisted Coding

Name : Vineeth-Chidurala **Date** : 16-01-2026

Ht.No. : 2303A52447

Task - 1 : Refactoring Odd/Even Logic (List Version)
Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.

Prompt : 1.Generate a program to calculate the sum of odd and even numbers in a list.

2. Generate a program to calculate the sum of odd and even numbers in a list

Screenshots :

```
#Generate a program to calculate the sum of odd and even numbers in a list.
nums = list(map(int, input("Enter numbers separated by space: ").split()))

even_sum = 0
odd_sum = 0

for n in nums:
    if n % 2 == 0:
        even_sum += n
    else:
        odd_sum += n

print("Sum of Even Numbers:", even_sum)
print("Sum of Odd Numbers:", odd_sum)
```

```
#Generate a program to calculate the sum of odd and even numbers in a list
nums = [1, 2, 3, 4, 5, 6]

even_sum = 0
odd_sum = 0

for n in nums:
    if n % 2 == 0:
        even_sum += n
    else:
        odd_sum += n

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

Output :

```
        if n % 2 == 0:
            even_sum += n
        else:
            odd_sum += n

    print("Sum of Even Numbers:", even_sum)
    print("Sum of Odd Numbers:", odd_sum)
```

[1] ✓ 2.4s

```
Sum of Even Numbers: 10
Sum of Odd Numbers: 0
```

```
    for n in nums:
        if n % 2 == 0:
            even_sum += n
        else:
            odd_sum += n

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

[2] ✓ 0.0s

```
Even Sum: 12
Odd Sum: 9
```

Task – 2 : Area Calculation Explanation

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

Prompt : Scenario: You are onboarding a junior developer. explain a func that calculates the area of different shapes. Expected Output: Code Explana on

Scenario: You are onboarding a junior developer. explain a func that calculates the area of different shapes. Expected Output: Code Explana on

ScreenShots :

```python
#Scenario: You are onboarding a junior developer. explain a func on that calculates the area of

def area(shape, *v):
    if shape == "circle":
        return 3.14 * v[0] * v[0]
    if shape == "rectangle":
        return v[0] * v[1]
    if shape == "triangle":
        return 0.5 * v[0] * v[1]

print(area("circle", 5))
print(area("rectangle", 4, 6))
print(area("triangle", 3, 8))
```

```python
#Scenario: You are onboarding a junior developer. explain a func that calculates the area of different

def area(shape, *v):
    if shape == "circle":
        return 3.14 * v[0] * v[0]
    elif shape == "rectangle":
        return v[0] * v[1]
    elif shape == "triangle":
        return 0.5 * v[0] * v[1]
    else:
        return "Invalid shape"

print(area("circle", 5))
print(area("rectangle", 4, 6))
print(area("triangle", 3, 8))
```

Output :

```python
    if shape == "rectangle":
        return v[0] * v[1]
    if shape == "triangle":
        return 0.5 * v[0] * v[1]

print(area("circle", 5))
print(area("rectangle", 4, 6))
print(area("triangle", 3, 8))
```

[3] ✓ 0.0s

```
78.5
24
12.0
```

```
    elif shape == "triangle":
        return 0.5 * v[0] * v[1]
    else:
        return "Invalid shape"

print(area("circle", 5))
print(area("rectangle", 4, 6))
print(area("triangle", 3, 8))
```

[4] ✓ 0.0s

```
78.5
24
12.0
```

Task-3 : Prompt Sensitivity Experiment

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

Code Style Variation :

1. Gemini

Uses single functions with conditional logic (if-elif) to handle multiple cases

Focuses on compact code and minimal structure.

Easy to read for beginners because everything is in one place.

Example concept :

```
def
  calculate_area(sha
  pe, **kwargs): if
  shape == "circle":

  return math.pi * kwargs["radius"] ** 2
```

✔ Simple
✔ Beginner-friendly
✘ Less modular

Cursor AI :

Uses separate functions for each task.

Follows modular programming principles.

Each function has a single responsibility.

Example concept :

```
def
  calculate_circle_ar
  ea(radius): return
  math.pi * radius **
  2
```

Explanation :

Gemini generates concise and beginner-friendly code that is easy to understand and suitable for small tasks and learning purposes. Cursor AI, however, produces more structured, modular, and robust code with proper error handling and extensibility, making it more suitable for real-world and production-level applications.

Task-4 : Tool Comparison Reflection

Based on your work in this topic, compare Gemini, Copilot, and Cursor AI

for usability and code quality.

Code :

```
def print_tool_comparison_reflection():
reflection =
```

Tool Comparison Reflection : Gemini vs Cursor AI

Based on the given outputs, Gemini is more beginner-friendly in terms of usability.

Its code is concise, readable, and includes clear docstrings and simple examples, making it suitable for quick learning and straightforward tasks.

However, its error handling is basic and the solutions are limited to the exact problem scope.

On the other hand, Cursor AI produces more structured and production-level code.

It separates logic into well-defined functions, includes proper validation using exceptions, and supports extensibility by handling more cases and shapes.

Although the code is more verbose, the overall code quality is higher and better suited for real-world applications.

Recommendation :

Gemini is ideal for beginners and rapid understanding, while Cursor AI is better for writing robust, maintainable, and scalable code.

print(reflection) # Call the function

print_tool_comparison_reflection()

Output :

```
Tool Comparison Reflection: Gemini vs Cursor AI

Based on the given outputs, Gemini is more beginner-friendly in terms of usability.
Its code is concise, readable, and includes clear docstrings and simple examples,
making it suitable for quick learning and straightforward tasks. However, its
error handling is basic and the solutions are limited to the exact problem scope.

On the other hand, Cursor AI produces more structured and production-level code.
It separates logic into well-defined functions, includes proper validation using
exceptions, and supports extensibility by handling more cases and shapes. Although
the code is more verbose, the overall code quality is higher and better suited for
real-world applications.

Recommendation:
Gemini is ideal for beginners and rapid understanding, while Cursor AI is better
for writing robust, maintainable, and scalable code.
```