

## Assignment – 2.5

AI Assisted Coding

Name: G SAIANIRUDH

Roll Number: 2303A51891

Batch - 04

16-01-2026

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

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure with files like task1.py, task2.py, task3.py, task5\_iterative.py, and task5\_recursive.py.
- Code Editor:** The main editor shows the original legacy-style code:

```
task1.py - AI-A-coding-v2 - Cursor
task1.py lines (1-9)

# Task 1: Refactoring Odd/Even Logic (List Version)
# Scenario:
# You are improving legacy code.
# 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

# Original Code (Legacy Style)
def calculate_sums_original(numbers):
    odd_sum = 0
    even_sum = 0
    i = 0
    while i < len(numbers):
        if numbers[i] % 2 == 0:
            even_sum = even_sum + numbers[i]
        else:
            odd_sum = odd_sum + numbers[i]
        i = i + 1
    return odd_sum, even_sum

# Test the original code
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd, even = calculate_sums_original(numbers)
print("Original code:")
print(f"Sum of odd numbers: {odd}")
print(f"Sum of even numbers: {even}")
```
- Output Panel:** Shows the terminal output of running the script:

```
PS E:\sem6\AI-A-coding-v2> & "c:\Python314\python.exe" "c:\Users\spurush\cursor\extensions\ms-python.debugger-2025.18.0\win32-x64\bundled\libs\debug\launcher" "5B401" -- "e:\sem6\AI-A-coding-v2\Assessment-2.5\task1.py"
Original Code:
Sum of odd numbers: 25
Sum of even numbers: 30
PS E:\sem6\AI-A-coding-v2>
```
- Refactor Preview:** A sidebar titled "Task1.py lines (1-9)" shows the refactored code. It includes the original code block and the improved/refactored code block.
- Improved/Refactored Code:** The sidebar also shows the improved/refactored code:

```
# Improved Code (Refactored)
def calculate_sums_improved(numbers):
    """
    Calculate the sum of odd and even numbers in a list.
    """
    odd_sum = 0
    even_sum = 0
    for number in numbers:
        if number % 2 == 0:
            even_sum += number
        else:
            odd_sum += number
    return odd_sum, even_sum
```

```

File Edit Selection View Go Run Terminal Help
task1.py U task1-2.py X
Assessment2.5 > task1-2.py > ...
1 # Improved Code (Refactored)
2 def calculate_sums_improved(numbers):
3     """
4         Calculate the sum of odd and even numbers in a list.
5     Args:
6         numbers: List of integers
7     Returns:
8         tuple: (sum_of_odd_numbers, sum_of_even_numbers)
9     """
10    odd_sum = sum(num for num in numbers if num % 2 != 0)
11    even_sum = sum(num for num in numbers if num % 2 == 0)
12
13    return odd_sum, even_sum
14
15 # Alternative: Improved version using filter.
16 def calculate_sums_alternative(numbers):
17     """Alternative refactored version using filter."""
18     odd_sum = sum(filter(lambda x: x % 2 != 0, numbers))
19     even_sum = sum(filter(lambda x: x % 2 == 0, numbers))
20
21     return odd_sum, even_sum
22
23 # Test the improved code
24 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
25 odd, even = calculate_sums_improved(numbers)
26 print("Improved Code:")
27 print(f"Sum of odd numbers: {odd}")
28 print(f"Sum of even numbers: {even}")
29
30 # Test alternative version
31 odd2, even2 = calculate_sums_alternative(numbers)
32 print("Alternative Improved Code:")
33 print(f"Sum of odd numbers: {odd2}")
34 print(f"Sum of even numbers: {even2}")

```

```

# Improved Code (Refactored)
def calculate_sums_improved(numbers):
    """
        Calculate the sum of odd and even numbers in a list.
    Args:
        numbers: List of integers
    Returns:
        tuple: (sum_of_odd_numbers, sum_of_even_numbers)
    """
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    even_sum = sum(num for num in numbers if num % 2 == 0)
    return odd_sum, even_sum

# Alternative Improved version using filter
def calculate_sums_alternative(numbers):
    """Alternative refactored version using filter."""
    odd_sum = sum(filter(lambda x: x % 2 != 0, numbers))
    even_sum = sum(filter(lambda x: x % 2 == 0, numbers))
    return odd_sum, even_sum

# Test the improved code
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd, even = calculate_sums_improved(numbers)
print("Improved Code:")
print(f"Sum of odd numbers: {odd}")
print(f"Sum of even numbers: {even}")

# Test alternative version
odd2, even2 = calculate_sums_alternative(numbers)
print("Alternative Improved Code:")
print(f"Sum of odd numbers: {odd2}")
print(f"Sum of even numbers: {even2}")

Improvements:
1. Readability: Uses list comprehensions and built-in sum().
2. Conciseness: Fewer lines, clearer intent.
3. Pythonic: Uses generator expressions and functional patterns.
4. Documentation: Adds a docstring.
5. Efficiency: Similar performance, more idiomatic.

Both versions produce the same output:
• Sum of odd numbers: 25 (1+3+5+7+9)
• Sum of even numbers: 30 (2+4+6+8+10)

Should I combine these into a single file, or do you want to test them separately?

```

Problems Output Debug Console Terminal Ports Python Debug Console + ...

E:\sem6\AI-A-coding-v2> C:
PS E:\sem6\AI-A-coding-v2> e: cd 'e:\sem6\AI-A-coding-v2'; & "c:\Python314\python.exe" 'c:\Users\spicus\cursor\extensions\ms-python-ls.0-win32-x64\bundled\libs\debugpy\launcher' '61638' '--' 'e:\sem6\AI-A-coding-v2\Assessment2.5\task1-2.py'

OUTLINE THUMBLINE

Cursor tab Ln 34 Col 39 Space: 4 UTF-8 GBLF Python Python 3.14 (64-bit) Go Live

## 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

The screenshot shows the GEMIN AI-CODE ASSIST interface. A code editor window displays Python code for calculating the area of various shapes. The code uses a function `calculate_area` with keyword arguments. A tooltip from the AI-assistant provides detailed information about the function's parameters and logic. Below the code editor is a preview pane showing the output of the code execution.

```

def calculate_area(shape, **kwargs):
    """
    For 'rectangle', expects 'length' and 'width'.
    For 'circle', expects 'radius'.
    For 'triangle', expects 'base' and 'height'.

    Returns:
        float: The calculated area of the shape.
        str: An error message if the shape is not supported or dimensions are missing.
    """
    if shape == "rectangle":
        if "length" in kwargs and "width" in kwargs:
            return kwargs["length"] * kwargs["width"]
        else:
            return "Error: Missing length or width for rectangle."
    elif shape == "circle":
        if "radius" in kwargs:
            return math.pi * (kwargs["radius"] ** 2)
        else:
            return "Error: Missing radius for circle."
    elif shape == "triangle":
        if "base" in kwargs and "height" in kwargs:
            return 0.5 * kwargs["base"] * kwargs["height"]
        else:
            return "Error: Missing base or height for triangle."
    else:
        return "Error: Unsupported shape."

# Example Usage
rectangle_area = calculate_area("rectangle", length=10, width=5)
print(f"The area of the rectangle is: {rectangle_area}")

circle_area = calculate_area("circle", radius=5)
print(f"The area of the circle is: {circle_area}")

triangle_area = calculate_area("triangle", base=8, height=4)
print(f"The area of the triangle is: {triangle_area}")

# Example of an error
square_area = calculate_area("square", side=5)
print(f"The area of the square is: {square_area}")

Caution: Tools can potentially perform harmful actions to your code and environment.
Learn more
    
```

Take action above to proceed

1 context item > Agent: Preview

This screenshot shows a more detailed view of the GEMIN AI-CODE ASSIST interface. It includes an explanation of the code, a function definition section with annotations, and a conditional logic section. The bottom part of the screen shows the terminal output where the code is run and its results.

**Explanation:**

This Python script defines a single function `calculate_area` that is designed to be a flexible tool for finding the area of several different geometric shapes.

**Function Definition:**

- `def calculate_area(shape, **kwargs)`
  - This defines a function named `calculate_area` that accepts two arguments:
    - `shape`: A string that tells the function which shape's area to calculate (e.g., "rectangle").
    - `**kwargs`: This is a special Python feature that allows the function to accept any number of keyword arguments. The length of `kwargs`'s three arguments are collected into a dictionary named `kwargs`. This makes the function versatile, as different shapes often require different parameters for area calculation.

**Conditional Logic:**

The function uses `if`, `elif` (else if), and `else` statements to determine which shape has been requested and then performs the correct calculation.

**Example Usage:**

```

print("The area of the rectangle is: {rectangle_area}")
print("The area of the circle is: {circle_area}")
print("The area of the triangle is: {triangle_area}")
print("The area of the square is: {square_area}")
    
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PAGES GITLOG

```

PS C:\Users\AI-CODE\Assessment2\> python calculate_area.py
The area of the rectangle is: 50
The area of the circle is: 78.54
The area of the triangle is: 16.0
The area of the square is: Error: Unsupported shape.
    
```

## Explanation

This Python script defines a single function `calculate_area` that is designed to be a flexible tool for finding the area of several different geometric shapes.

### Function Definition:

```
def calculate_area(shape, **kwargs):
```

This defines a function named `calculate_area` that accepts two arguments:

`shape`: A string that tells the function which shape's area to calculate (e.g., "rectangle").

**\*\*kwargs:** This is a special Python feature that allows the function to accept any number of keyword arguments (like `length=10` or `radius=5`). These arguments are collected into a dictionary named `kwargs`. This makes the function adaptable, as different shapes require different parameters for area calculation.

Conditional Logic:

The function uses `if`, `elif` (else if), and `else` statements to determine which shape has been requested and then performs the correct calculation.

`if shape == "rectangle":`

If the shape is a "rectangle", the code checks if the necessary dimensions, `length` and `width`, are provided in the `kwargs` dictionary.

If they are present, it calculates the area by multiplying `length` and `width`.

If they are missing, it returns an error message.

`elif shape == "circle":`

If the shape is a "circle", it checks for a `radius`.

If found, it calculates the area using the formula  $\pi * r^2$ , importing the `math` module to get an accurate value of `math.pi`. If the `radius` is missing, it returns an error. `elif shape == "triangle":`

If the shape is a "triangle", it checks for a `base` and `height`.

If they are present, it calculates the area using the formula  $0.5 * base * height$ .

If they are missing, it returns an error.

`else:`

If the provided shape name does not match any of the supported shapes ("rectangle", "circle", "triangle"), the function returns an "Unsupported shape" error.

Example Usage:

The lines at the end of the script demonstrate how to use the function and print the results for a rectangle, a circle, a triangle, and an unsupported shape to show how the error handling works.

### 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

```
# Write a Python function to reverse an integer.
def reverse(x: int) -> int:
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31
    sign = -1 if x < 0 else 1
    x = abs(x)
    rev = 0
    while x != 0:
        rev = rev * 10 + x % 10
        x //= 10
    rev *= sign
    if rev < INT_MIN or rev > INT_MAX:
        return 0
    return rev
```

Terminal output:

```
PS E:\sem6\AI-A-coding-v2> & 'c:\Python314\python.exe' 'c:\Users\sprus\cursor\extensions\ms-python.debugger-2025.18.0-win32-x64\bundle\libs\debugger\launcher' '59870' '->' 'e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
PS E:\sem6\AI-A-coding-v2> 9547
9547
● PS E:\sem6\AI-A-coding-v2> cd 'e:\sem6\AI-A-coding-v2'; & 'c:\Python314\python.exe' 'c:\Users\sprus\cursor\extensions\ms-python.debugger-2025.18.0-win32-x64\bundle\libs\debugger\launcher' '62775' '->' 'e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
● PS E:\sem6\AI-A-coding-v2> cd 'e:\sem6\AI-A-coding-v2'; & 'c:\Python314\python.exe' 'c:\Users\sprus\cursor\extensions\ms-python.debugger-2025.18.0-win32-x64\bundle\libs\debugger\launcher' '62814' '->' 'e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
8520
○ PS E:\sem6\AI-A-coding-v2>
```



The screenshot displays three AI coding tools side-by-side, each showing the same Python code for reversing an integer with strict overflow checks. The code is as follows:

```

task3.py - AI-A-coding-v2 - Cursor
task3.py - AI-A-coding-v2 - Copilot
task3.py - Gemini

# Write an optimized Python solution for reversing an integer with strict overflow checks for coding interviews.
def reverse_integer(x):
    ...
    Time Complexity: O(log|x|) - where n is the input number
    Space Complexity: O(1) - constant extra space

    Args:
        x: An integer to reverse

    Returns:
        The reversed integer, or None if the reversed integer overflows 32-bit signed integer range
        (i.e., outside [-2^31, 2^31 - 1] = [-2147483648, 2147483647])
        ...
        # 32-bit signed integer bounds
        INT_MIN = 2**31 - 1 = 2147483648
        INT_MAX = 2**31 - 1 = 2147483647

    result = 0
    ...
    while x != 0:
        digit = x % 10 # Extract the last digit
        digit = x // 10 # Handle negative modulo correctly
        x = x // 10 # Handle negative division correctly
        ...
        # Strict overflow check BEFORE multiplication
        # Check if result * 10 + digit would overflow
        ...
        # For positive overflow: result * 10 + digit > INT_MAX
        # If result == INT_MAX // 10 or (result == INT_MAX // 10 and digit < INT_MIN % 10):
            return None
        ...
        # Safe to perform the operation
        result = result * 10 + digit
        ...
    return result

# Alternative optimized version (more Pythonic and cleaner)
def reverse_integer_v2(x):
    ...
    Alternative optimized solution - cleaner approach handling sign separately.
    Time Complexity: O(log|x|)

```

**Cursor AI (Left):**

- Output: `task3.py`
- Terminal: `task3.py`
- PowerShell Extension v2025.4.0
- Copy/Cut/Copy/Paste
- https://www.microsoft-powershell.com
- Type "help" to get help.
- PS E:\vse\vscode\AI-A-coding> [ ]

**Copilot (Middle):**

- Output: `task3.py`
- Terminal: `task3.py`
- PowerShell Extension v2025.4.0
- Copy/Cut/Copy/Paste
- https://www.microsoft-powershell.com
- Type "help" to get help.
- PS E:\vse\vscode\AI-A-coding> [ ]

**Gemini (Right):**

- Output: `task3.py`
- Terminal: `task3.py`
- PowerShell Extension v2025.4.0
- Copy/Cut/Copy/Paste
- https://www.microsoft-powershell.com
- Type "help" to get help.
- PS E:\vse\vscode\AI-A-coding> [ ]

## Task 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

Based on my experience using Gemini, GitHub Copilot, and Cursor AI during this topic, I observed clear differences in both usability and code quality.

Gemini is useful for understanding concepts and generating explanations, but it often produces generic code unless very strict constraints are provided. It is better suited for learning and problem understanding rather than competitive or production-level coding.

GitHub Copilot integrates smoothly with IDEs like VS Code and provides fast, context-aware code suggestions. However, its outputs sometimes assume the developer will handle edge cases, so overflow handling and constraints may be missed unless explicitly guided.

Cursor AI provided the best balance of usability and code quality. It allows direct interaction with the codebase, understands existing files, and responds well to detailed prompts. When constraints are clearly mentioned, Cursor AI consistently generated correct, optimized, and readable code, making it ideal for real development and debugging tasks.

Conclusion:

For learning → Gemini

For quick coding assistance → Copilot

For serious development and prompt-based experimentation → Cursor AI