

Assignment – 2.5

Name: N. Vishnu Vardhan Reddy

Roll Number: 2303A51210

Batch - 04

AI Assisted Coding

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 code in `task1.py`.

```
# 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

# 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("Sum of odd numbers: (odd)")
print("Sum of even numbers: (even)")

# Test the refactored code
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd, even = calculate_sums_improved(numbers)
print("Refactored code:")
print("Sum of odd numbers: (odd)")
print("Sum of even numbers: (even)")
```
- Terminal:** Shows the command run in the terminal: `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' '58401' '--' 'E:\sem6\AI-A-coding-v2\Assessment2.5\task1.py'`. The output shows the original and refactored code results.
- Output:** Shows the results of the code execution: "Sum of odd numbers: 25" and "Sum of even numbers: 30" for the original code, and "Sum of odd numbers: 25" and "Sum of even numbers: 30" for the refactored code.
- Status Bar:** Shows the cursor position (Line 28, Col 38), spaces (4), and Python version (Python 3.14 (64-bit)).

```

task1-2.py - AI-A-coding-v2 - Cursor
File Edit Selection View Go Run Terminal Help
task1.py 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
6         Args:
7             numbers: List of integers
8
9         Returns:
10            tuple: (sum_of_odd_numbers, sum_of_even_numbers)
11        """
12
13    odd_sum = sum(num for num in numbers if num % 2 != 0)
14    even_sum = sum(num for num in numbers if num % 2 == 0)
15
16    return odd_sum, even_sum
17
18 # Alternative Improved version using filter
19 #def calculate_sums_alternative(numbers):
20 #    """Alternative refactored version using filter."""
21 #    odd_sum = sum(filter(lambda x: x % 2 != 0, numbers))
22 #    even_sum = sum(filter(lambda x: x % 2 == 0, numbers))
23 #    return odd_sum, even_sum
24
25 # Test the improved code
26 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
27 odd, even = calculate_sums_improved(numbers)
28 print("Improved Code:")
29 print(f"Sum of odd numbers: {odd}")
30 print(f"Sum of even numbers: {even}")
31
32 # Test alternative version
33 odd2, even2 = calculate_sums_alternative(numbers)
34 print("Alternative Improved Code:")
35 print(f"Sum of odd numbers: {odd2}")
36 print(f"Sum of even numbers: {even2}")
37
38 print("Sum of odd numbers: (odd2) ")
39
40 print("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(filter(lambda x: x % 2 != 0, numbers))
    even = sum(filter(lambda x: x % 2 == 0, numbers))
    return odd, even

# 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 (0+2+4+6+8+10)

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

```

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 Gemini AI interface with a code editor window titled "Q AI-A-coding-v2". The code is a Python script for calculating the area of different shapes. It defines a function `calculate_area` that takes a shape name and keyword arguments for dimensions. The script handles four shapes: rectangle, circle, triangle, and square. It includes example usage and error handling for unsupported shapes. A tooltip from "Gemini Code Assist" provides information about the current code segment.

```
File Edit Selection View Go Run Terminal Help Q AI-A-coding-v2
Gemini CODE ASSIST CHAT
Preview Channel
task2.py x
Assessment25 > task2py
1 Press Ctrl+I to use Gemini Code Assist to help you code. Start typing to dismiss.

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("The area of the rectangle is: (rectangle_area)")

circle_area = calculate_area("circle", radius=7)
print("The area of the circle is: (circle_area)")

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

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

Gemini Tools can potentially perform harmful actions to your code and environment.
Learn more

Reject Accept Edit
Generated by Gemini gemini-2.5-pro

Take action above to proceed.
1 context item > Agent Preview
16 main 21 Et SproutCCoder1 needs your review 0 0 △
In 1, Col 1 Spacing: 4 UTT-B CLF ⓘ Python 3.14.0 ⓘ Go Live +
```

The screenshot shows the Gemini AI interface with a code editor window titled "Q AI-A-coding-v2". The code is the same Python script for calculating areas. This time, the "Explanation" section is expanded, providing a detailed breakdown of the code's logic and behavior. It explains the function definition, conditional logic for shape recognition, and the calculation logic for each shape type. The explanation also covers error handling and example usage.

```
File Edit Selection View Go Run Terminal Help Q AI-A-coding-v2
Gemini CODE ASSIST CHAT
Preview Channel
task2.py x
Assessment25 > task2py
1 import math
2
3 def calculate_area(shape, **kwargs):
4     ...
5     Calculates the area of different shapes.
6
7     Args:
8         shape (str): The name of the shape (e.g., 'rectangle', 'circle', 'triangle').
9         **kwargs: The dimensions of the shape.
10            For 'rectangle', expects 'length' and 'width'.
11            For 'circle', expects 'radius'.
12            For 'triangle', expects 'base' and 'height'.
13
14     Returns:
15         - float: The calculated area of the shape.
16         - str: An error message if the shape is not supported or dimensions are missing.
17
18     If shape == "rectangle":
19         if "length" in kwargs and "width" in kwargs:
20             return kwargs["length"] * kwargs["width"]
21         else:
22             return "Error: Missing length or width for rectangle."
23     elif shape == "circle":
24         if "radius" in kwargs:
25             return math.pi * (kwargs["radius"] ** 2)
26         else:
27             return "Error: Missing radius for circle."
28     elif shape == "triangle":
29         if "base" in kwargs and "height" in kwargs:
30             return 0.5 * kwargs["base"] * kwargs["height"]
31         else:
32             return "Error: Missing base or height for triangle."
33     else:
34         return "Error: Unsupported shape."
35
36 # Example Usage:
37 rectangle_area = calculate_area("rectangle", length=10, width=5)
38 print("The area of the rectangle is: (rectangle_area)")
39
40 circle_area = calculate_area("circle", radius=7)
41 print("The area of the circle is: (circle_area)")
42
43 triangle_area = calculate_area("triangle", base=8, height=4)
44 print("The area of the triangle is: (triangle_area)")
45
46 # Example of an error:
47 square_area = calculate_area("square", side=5)
48 print("The area of the square is: (square_area)")

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.
- If shape == "circle":
  - If the shape is a 'circle', it checks for a radius.
  - If it is present, 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.
- If 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  $0.5 \times \text{base} \times \text{height}$ .
  - 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:
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 handing works.

Generated by Gemini gemini-2.5-pro
```

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 $0.5 * \text{base} * \text{height}$.

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**

The screenshot shows a code editor interface with several tabs open, each containing Python code. The tabs include 'task1.py', 'task1-2.py', 'task2.py', and 'task3.py'. The 'task3.py' tab is currently active, displaying the following code:

```
# 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
```

Below the code editor is a terminal window showing command-line history. The terminal output includes:

```
PS E:\sem\VAI-A-coding-v2> & "c:\Python\14\python.exe" "c:\Users\sprusu.cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher" 59878 ... "e:\sem\VAI-A-coding-v2\Assessment2.5\task3.py"
PS E:\sem\VAI-A-coding-v2>
PS E:\sem\VAI-A-coding-v2> cd "e:\sem\VAI-A-coding-v2" & "c:\Python\14\python.exe" "c:\Users\sprusu.cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher" 62775 ... "e:\sem\VAI-A-coding-v2\Assessment2.5\task3.py"
PS E:\sem\VAI-A-coding-v2> cd "e:\sem\VAI-A-coding-v2" & "c:\Python\14\python.exe" "c:\Users\sprusu.cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher" 62814 ... "e:\sem\VAI-A-coding-v2\Assessment2.5\task3.py"
PS E:\sem\VAI-A-coding-v2>
```

The terminal also shows a prompt: 'Ps E:\sem\VAI-A-coding-v2>'. The status bar at the bottom indicates the current file is 'task3.py'.

The image shows two side-by-side instances of the Visual Studio Code (VS Code) code editor. Both instances have the title bar "task3.py - AI-A-coding-v2 - Cursor".

Left Window (task3.py):

- Code:** The code implements a function `reverse_integer` that reverses an integer while handling 32-bit signed integer overflow. It uses a loop to extract digits and handle negative numbers.
- Output:** The terminal shows the code running and outputting results for various test cases, including edge cases like 32-bit overflow and zero.
- Terminal:** The PowerShell Extension v205.4.8 is used to run the script.

Right Window (task3.py):

- Code:** The same `reverse_integer` function, but with additional comments explaining complexity and space requirements.
- Output:** The terminal shows the code running and outputting results for various test cases.
- Terminal:** The PowerShell Extension v205.4.8 is used to run the script.

Both windows show the status bar at the bottom with file paths like "E:\akademie\AI-A-coding-v2\Assessment2.5\task3.py" and "E:\akademie\AI-A-coding-v2\Assessment2.5\task3.py". The right window also shows a "Review" tab and a "Review" button in the bottom right corner.

```

File Edit Selection View Go Run Terminal Help task3.3.py - AI-A-coding-v2 - Cursor
task3.3.py
task1.py task2.py task3.py task4.py
Assessment2.5 task3.3.py ~
1 # Write an optimized Python solution for reversing an integer with strict overflow checks for coding interviews.
2
3 def reverse_integer(x):
4     """
5         Optimized solution to reverse an integer with strict 32-bit signed integer overflow handling.
6
7             Time Complexity: O(log|N|) - where n is the input number
8             Space Complexity: O(1) - constant extra space
9
10            Args:
11                x: An integer to reverse
12
13            Returns:
14                The reversed integer, or 0 if the reversed integer overflows 32-bit signed integer range
15                (i.e., outside [-2^31, 2^31 - 1] = [-2147483648, 2147483647])
16
17            Raises:
18                ValueError: If the input is not a valid 32-bit signed integer boundary.
19                INT_MIN = -2**31 = -2147483648
20                INT_MAX = 2**31 - 1 = 2147483647
21
22            result = 0
23
24            while x != 0:
25                # Extract the last digit
26                digit = x % 10 if x > 0 else -(x // 10) # Handle negative modulo correctly
27                x = x // 10 if x > 0 else -x // 10 # Handle negative division correctly
28
29                # Strict overflow check BEFORE multiplication
30                # Check if result * 10 would overflow
31                # For positive overflow: result * 10 + digit > INT_MAX
32                # Negative result <= INT_MIN - digit / 10
33                if result > INT_MAX // 10 or (result == INT_MAX // 10 and digit > INT_MAX % 10):
34                    return 0
35
36                # For negative overflow: result * 10 + digit < INT_MIN
37                # Negative result > (INT_MIN - digit) / 10
38                if result < INT_MIN // 10 or (result == INT_MIN // 10 and digit < INT_MIN % 10):
39                    return 0
40
41                # Safe to perform the operation
42                result = result * 10 + digit
43
44            return result
45
46
47 # Alternative optimized version (more Pythonic and cleaner)
48 def reverse_integer_v2(x):
49     """
50         Alternative optimized solution - cleaner approach handling sign separately.
51
52             Time Complexity: O(log|N|)
53
PowerShell Extension V2025.4.0
Copyright (c) Microsoft Corporation.
https://aka.ms/vscode-powershell
Type 'help' to get help.

PS E:\seme\AI-A-coding-v2 []

```

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