

Assignment - 2.5

Name: K.Sathvik Reddy

Roll Number: 2303A51202

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 .github, .vscode, Assessment2.5, task1.py, Assignment-1.5, and task5_recursive.py.
- Code Editor:** The main editor shows the original code in Legacy Style:

```
1 # Task 1: Refactoring Odd/Even Logic (List Version)
2 # Scenario:
3 # You are improving legacy code.
4 # Task:
5 # Write a program to calculate the sum of odd and even numbers in a list,
6 # then refactor it using AI.
7 # Expected Output:
8 # Original and Improved code
9
10 # Original Code (Legacy Style)
11 def calculate_sums_original(numbers):
12     odd_sum = 0
13     even_sum = 0
14     i = 0
15     while i < len(numbers):
16         if numbers[i] % 2 == 0:
17             even_sum = even_sum + numbers[i]
18         else:
19             odd_sum = odd_sum + numbers[i]
20         i = i + 1
21     return odd_sum, even_sum
22
23 # Test the original code
24 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
25 odd, even = calculate_sums_original(numbers)
26 print("Original Code:")
27 print(f"Sum of odd numbers: {odd}")
28 print(f"Sum of even numbers: {even}")
```
- Output Panel:** Shows the terminal output of the original code execution:

```
PS E:\sem6\AI-A-coding-v2 & "c:\Python314\python.exe" "c:\Users\sprus\cursor\extensions\vs-pyhon.debug-2025.18.0-win32-x64\bundled\libs\debug\launcher" '58401' '--' 'E:\sem6\AI-A-coding-v2\Assessment2.5\Task1.py'
● Original Code:
Sum of odd numbers: 25
Sum of even numbers: 30
○ PS E:\sem6\AI-A-coding-v2>
```
- Refactor View:** A floating window titled "Task1.py lines 1-9" displays the refactored code:

```
# 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(f"Sum of odd numbers: {odd}")
print(f"Sum of even numbers: {even}")
```
- Refactor View (Bottom):** A floating window titled "Improved/Refactored Code:" shows the improved code:

```
# Improved Code (Refactored)
def calculate_sums_improved(numbers):
    """
    Calculate the sum of odd and even numbers in a list.
    """
    Plan, @ for context, / for commands
    Ask Auto 1x
```

```

task1.py - AI-A-coding-v2 - Cursor
task1.py U task1-2.py U
File Edit Selection View Go Run Terminal Help
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    odd_sum = sum(num for num in numbers if num % 2 != 0)
13    even_sum = sum(num for num in numbers if num % 2 == 0)
14    return odd_sum, even_sum
15
16 # Alternative Improved version using filter
17 def calculate_sums_improved(numbers):
18     """
19         Alternative refactored version using filter.
20     """
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 # Test alternative version
39 odd, even = calculate_sums_improved(numbers)
40 print("Improved Code:")
41 print(f"Sum of odd numbers: {odd}")
42 print(f"Sum of even numbers: {even}")
43
44 odd, even = calculate_sums_improved(numbers)
45 print("Alternative Improved Code:")
46 print(f"Sum of odd numbers: {odd}")
47 print(f"Sum of even numbers: {even}")

task1.py lines 1-9
task1.py (1-9)
# 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_improved(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
odd, even = calculate_sums_improved(numbers)
print("Alternative Improved Code:")
print(f"Sum of odd numbers: {odd}")
print(f"Sum of even numbers: {even}")

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?

```

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

```

GEMINI CODE ASSIST:CHAT
File Edit Selection View Go Run Terminal Help ← →
Preview Channel
task2.py
Assessment2.5 > task2.py
1 Press Ctrl+K to ask Gemini Code Assist to help your 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 math.pi * 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=5, width=3)
print(f"The area of the rectangle is: {rectangle_area}")

circle_area = calculate_area("circle", radius=3)
print(f"The area of the circle is: {circle_area}")

triangle_area = calculate_area("triangle", base=6, 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}")

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

Reject Accept Edit
Generated by Gemini (version 2.5-pro)

Take action above to proceed
1 context item Agent Preview

```

```

GEMINI CODE ASSIST:CHAT
File Edit Selection View Go Run Terminal Help ← →
task2.py
Assessment2.5 > task2.py ...
import math

def calculate_area(shape, **kwargs):
    """
    Calculates the area of different shapes.
    """

    Args:
        shape (str): The name of the shape (e.g., 'rectangle', 'circle', 'triangle').
        **kwargs: The dimensions of the shape.
            - 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 math.pi * 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=5, width=3)
print(f"The area of the rectangle is: {rectangle_area}")

circle_area = calculate_area("circle", radius=3)
print(f"The area of the circle is: {circle_area}")

triangle_area = calculate_area("triangle", base=6, height=4)
print(f"The area of the triangle is: {triangle_area}")

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

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):
    o 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: A variable number of keyword arguments that allow the function to accept any number of keyword arguments (e.g., 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, else if, and else statements to determine which shape has been requested and then performs the correct calculation.

• if shape == "rectangle":
    o If the shape is a "rectangle", the code checks if the necessary dimensions, [length] and [width], are provided in the kwargs dictionary.
    o If they are present, it calculates the area by multiplying length and width.
    o If they are missing, it returns an error message.
• elif shape == "circle":
    o If the shape is a "circle", it checks for a radius.
    o If found, it calculates the area using the formula  $\pi r^2$ , importing the math module to get an accurate value of [math.pi].
    o If the radius is missing, it returns an error.
• elif shape == "triangle":
    o If the shape is a "triangle", it checks for a base and height.
    o If they are present, it calculates the area using the formula  $0.5 \times \text{base} \times \text{height}$ .
    o If they are missing, it returns an error.
• else:
    o 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 (version 2.5-pro)

Ask Gemini or type 'b'
1 context item Agent Preview

```

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.

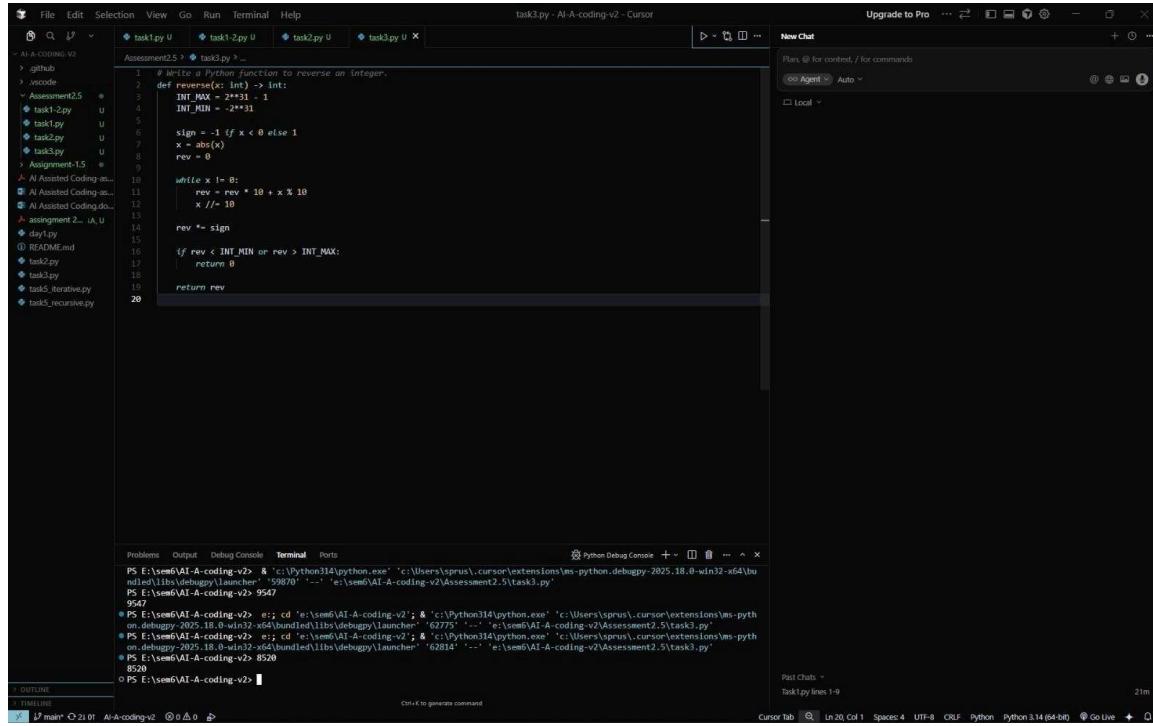
❖ Task:

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 + 1
    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
```

The screenshot shows a code editor interface with two tabs open: task1.py and task3.py. The task3.py tab is active, displaying the provided code. The code is a Python function named reverse that takes an integer x and returns its reverse. It handles overflow by checking against INT_MAX and INT_MIN. The code uses a while loop and integer division (//) for reversing the number. A cursor is visible at the end of the code. Below the code editor is a terminal window showing the command PS E:\sem6\AI-A-coding-v2> & c:\Python314\python.exe 'c:\Users\sprous\cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bin\debugpy\1.1.0\Windows\Launcher' '59870' --> e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py. The terminal output includes several error messages related to file paths and permissions, indicating that the script is trying to run but failing due to access issues.


```

task3.3.py - AI-A-coding-v2 - Cursor
File Edit Selection View Go Run Terminal Help
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    # 32-bit signed integer bounds
18    INT_MIN = -2**31 # -2147483648
19    INT_MAX = 2**31 - 1 # 2147483647
20
21    result = 0
22
23    while x != 0:
24        digit = abs(x % 10) # Get the last digit
25        digit = digit * 10 if x < 0 else -(digit % 10) # Handle negative modulo correctly
26        x = x // 10 if x > 0 else -(x // 10) # Handle negative division correctly
27
28        # Strict overflow check BEFORE multiplication
29        # Check if result * 10 + digit would overflow
30
31        # For positive overflow: result * 10 + digit > INT_MAX
32        # If result == INT_MAX // 10 or (result == INT_MAX // 10 and digit > INT_MAX % 10):
33        #     return 0
34
35        # For negative overflow: result * 10 + digit < INT_MIN
36        # If result == INT_MIN // 10 or (result == INT_MIN // 10 and digit < INT_MIN % 10):
37        #     return 0
38
39        # Safe to perform the operation
40        result = result * 10 + digit
41
42    return result
43
44
45    # Alternative optimized version (more Pythonic and cleaner)
46    def reverse_integer_v2(x):
47        """
48            Alternative optimized solution - cleaner approach handling sign separately.
49
50            Time complexity: O(log(N))
51
52    Problems Output Debug Console Terminal Ports
PowerShell Extension v2023.4.0
Copyright (c) Microsoft Corporation.
https://aka.ms/vscode-powershell
Type 'help' to get help.

PS E:\vscode\AI-A-coding-v2\Assessment2.5\

+-----+
| OUTLINE | TIMELINE | V main* | 21:01 | AI-A-coding-v2 | ② ③ ④ ⑤ |
+-----+

```



```

task3.3.py - AI-A-coding-v2 - Cursor
File Edit Selection View Go Run Terminal Help
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    # 32-bit signed integer bounds
18    INT_MIN = -2**31 # -2147483648
19    INT_MAX = 2**31 - 1 # 2147483647
20
21    result = 0
22
23    while x != 0:
24        digit = abs(x % 10) # Get the last digit
25        digit = digit * 10 if x < 0 else -(digit % 10) # Handle negative modulo correctly
26        x = x // 10 if x > 0 else -(x // 10) # Handle negative division correctly
27
28        # Strict overflow check BEFORE multiplication
29        # Check if result * 10 + digit would overflow
30
31        # For positive overflow: result * 10 + digit > INT_MAX
32        # If result == INT_MAX // 10 or (result == INT_MAX // 10 and digit > INT_MAX % 10):
33        #     return 0
34
35        # For negative overflow: result * 10 + digit < INT_MIN
36        # If result == INT_MIN // 10 or (result == INT_MIN // 10 and digit < INT_MIN % 10):
37        #     return 0
38
39        # Safe to perform the operation
40        result = result * 10 + digit
41
42    return result
43
44
45    # Alternative optimized version (more Pythonic and cleaner)
46    def reverse_integer_v2(x):
47        """
48            Alternative optimized solution - cleaner approach handling sign separately.
49
50            Time complexity: O(log(N))
51
52    Problems Output Debug Console Terminal Ports
PowerShell Extension v2023.4.0
Copyright (c) Microsoft Corporation.
https://aka.ms/vscode-powershell
Type 'help' to get help.

PS E:\vscode\AI-A-coding-v2\Assessment2.5\

+-----+
| OUTLINE | TIMELINE | V main* | 21:01 | AI-A-coding-v2 | ② ③ ④ ⑤ |
+-----+

```

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