

Assignment – 2.5

Name: A. Akshith Reddy

Roll Number: 2303A51177

Batch - 03

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 under "AI-A-CODING-V2".
- Code Editor:** Displays the file "task1.py" with two sections: "Original Code (Legacy Style)" and "Improved/Refactored Code".
- Terminal:** Shows the command line output of running the script with sample input [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].
- Status Bar:** Shows the cursor position at Line 28, Column 38, and the file is saved.

```
# 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}")

# 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

# Test the refactored 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}")


```

```

task1-2.py - AI-A-coding-v2 - Cursor
File Edit Selection View Go Run Terminal Help
task1.py U task1-2.py U
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_alternative(numbers):
18     """Alternative refactored version using filter."""
19     odd, even = calculate_sums_improved(numbers)
20     even_sum = sum(filter(lambda x: x % 2 == 0, numbers))
21     return odd, even
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(f"\nImproved 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(f"\nAlternative Improved Code:")
33 print(f"Sum of odd numbers: {odd2}")
34 print(f"Sum of even numbers: {even2}")

task1.py lines 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_alternative(numbers):
    """Alternative refactored version using filter."""
    odd, even = calculate_sums_improved(numbers)
    even_sum = 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(f"\nImproved Code:")
print(f"Sum of odd numbers: {odd}")
print(f"Sum of even numbers: {even}")
# Test alternative version
odd, even = calculate_sums_alternative(numbers)
print(f"\nAlternative 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?

```

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

```
task2.py
1 Press Ctrl+I to ask Gemini Code Assist to help you code. Start typing to dismiss.
```

Hello, Akshith
How can Gemini help?

PROMPTS TO TRY

- How do I use Gemini Code Assist?
- Ask Gemini or type '@'

1 context item >

Default Agent Preview

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\ASUS\OneDrive\Desktop\AI Assisted Coding>

+ ... | ☰ x

powershell Python

Describe what to build next

Agent Auto ⌂ Go Live

Ln 1, Col 1 Spaces:4 UTF-8 CRLF () Python 3.12.10 (Microsoft Store) ⌂ Go Live

```
task2.py
1 Press Ctrl+I to ask Gemini Code Assist to help you code. Start typing to dismiss.
```

```
task2.py
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=6, 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)")

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

Take action above to proceed

1 context item >

Reject Accept ⌂ Edit

Generated by Gemini gemini-25-pro

+ ... | ☰ x

powershell Python

Describe what to build next

Agent Auto ⌂ Go Live

Ln 1, Col 1 Spaces:4 UTF-8 CRLF () Python 3.14.0 ⌂ Go Live

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. The active tab is 'task3.py' under the 'Assessment2.5' folder. The code in the editor is:

```
1 # Write a Python function to reverse an integer.
2 def reverse(x: int) -> int:
3     INT_MAX = 2**31 - 1
4     INT_MIN = -2**31
5
6     sign = -1 if x < 0 else 1
7     x = abs(x)
8     rev = 0
9
10    while x != 0:
11        rev = rev * 10 + x % 10
12        x //= 10
13
14    rev *= sign
15
16    if rev < INT_MIN or rev > INT_MAX:
17        return 0
18
19    return rev
```

Line 28 is highlighted.

Below the code editor is a terminal window showing a command-line session:

```
P5 E:\sem6\AI-A-coding-v2> 8 & "c:\Python314\python.exe" "c:\Users\sprous\cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher" "59870" -- "e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py"
PS E:\sem6\AI-A-coding-v2> 9547
■ P5 E:\sem6\AI-A-coding-v2> e; cd "e:\sem6\AI-A-coding-v2"; & "c:\Python314\python.exe" "c:\Users\sprous\cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher" "63775" -- "e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py"
■ P5 E:\sem6\AI-A-coding-v2> e; cd "e:\sem6\AI-A-coding-v2"; & "c:\Python314\python.exe" "c:\Users\sprous\cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher" "62814" -- "e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py"
■ P5 E:\sem6\AI-A-coding-v2> 8520
8520
■ P5 E:\sem6\AI-A-coding-v2>
```

task3.2.py - AI-A-coding-v2 - Cursor

```

File Edit Selection View Go Run Terminal Help
task1.py task2.py task2.py task3.py task3.py task3.2.py

Assessment2.5 * task3.2.py ~
def reverse_integer(x):
    """
    Reverse an integer with 32-bit signed integer overflow handling.
    """
    if x < 0:
        is_negative = True
        num = abs(x)
    else:
        is_negative = False
        num = x

    reversed_num = 0
    while num > 0:
        reversed_num = reversed_num * 10 + num % 10
        num //= 10

    # Check for 32-bit signed integer overflow
    INT_MIN = -2**31 - 1
    INT_MAX = 2**31 - 1
    if reversed_num < INT_MIN or reversed_num > INT_MAX:
        return 0

    return reversed_num

# Test cases
if __name__ == "__main__":
    # Test case 1: Positive number
    print(f"reverse_integer(123) = {reverse_integer(123)}") # Expected: 321
    # Test case 2: Negative number
    print(f"reverse_integer(-123) = {reverse_integer(-123)}") # Expected: -321
    # Test case 3: Number ending with zero
    print(f"reverse_integer(120) = {reverse_integer(120)}") # Expected: 21
    # Test case 4: Single digit
    print(f"reverse_integer(5) = {reverse_integer(5)}") # Expected: 5
    # Test case 5: Zero
    print(f"reverse_integer(0) = {reverse_integer(0)}") # Expected: 0
    # Test case 6: Overflow case (reversed number exceeds INT_MAX)
    print(f"reverse_integer(154236468) = {reverse_integer(154236468)}") # Expected: 0 (overflow)
    # Test case 7: Overflow case (reversed number exceeds INT_MIN)
    print(f"reverse_integer(-2147483648) = {reverse_integer(-2147483648)}") # Expected: 0 (overflow)

```

Up to Date

Integer reversal with overflow handling

Write a Python function to reverse an integer with 32-bit signed integer overflow handling.

Checking the current file for context.

Read task3.2.py

Writing a function to reverse an integer with 32-bit signed integer overflow handling.

Edit legal value for intellisense

Reading the file again.

Read task3.2.py

Improving overflow handling: checking during reversal to prevent intermediate values from exceeding 32-bit bounds.

Edit legal value for intellisense

Rewriting the file with improved overflow handling.

Edit legal value for intellisense

Checking the file and trying a different approach.

Read task3.2.py

Improving overflow handling by checking during reversal instead of after.

Edit legal value for intellisense

Testing the current implementation:

Run command: cd python
cd "e:\user\ai-a-coding-v2\assessment2.5" & python task3.2.py

Use Advisor? Skip Abort! + 1 or Run

Problems Output Debug Console Terminal Ports

PowerShell Extension v2025.4.0 Copyright (c) Microsoft Corporation. https://aka.ms/vscode-powershell Type 'help' to get help.

PS E:\user\ai-a-coding-v2

task3.2.py - AI-A-coding-v2 - Cursor

```

File Edit Selection View Go Run Terminal Help
task1.py task2.py task2.py task3.py task3.py task3.2.py

Assessment2.5 * task3.2.py ~
def reverse_integer(x):
    """
    Reverse an integer with 32-bit signed integer overflow handling.
    """
    if x < 0:
        is_negative = True
        num = abs(x)
    else:
        is_negative = False
        num = x

    reversed_num = 0
    while num > 0:
        reversed_num = reversed_num * 10 + num % 10
        num //= 10

    # Apply sign
    if is_negative:
        reversed_num = -reversed_num
    else:
        reversed_num = reversed_num

    # Check for 32-bit signed integer overflow
    INT_MIN = -2**31 - 1
    INT_MAX = 2**31 - 1
    if reversed_num < INT_MIN or reversed_num > INT_MAX:
        return 0

    return reversed_num

# Test cases
if __name__ == "__main__":
    # Test case 1: Positive number
    print(f"reverse_integer(123) = {reverse_integer(123)}") # Expected: 321
    # Test case 2: Negative number
    print(f"reverse_integer(-123) = {reverse_integer(-123)}") # Expected: -321
    # Test case 3: Number ending with zero
    print(f"reverse_integer(120) = {reverse_integer(120)}") # Expected: 21
    # Test case 4: Single digit
    print(f"reverse_integer(5) = {reverse_integer(5)}") # Expected: 5
    # Test case 5: Zero
    print(f"reverse_integer(0) = {reverse_integer(0)}") # Expected: 0
    # Test case 6: Overflow case (reversed number exceeds INT_MAX)
    print(f"reverse_integer(154236468) = {reverse_integer(154236468)}") # Expected: 0 (overflow)
    # Test case 7: Overflow case (reversed number exceeds INT_MIN)
    print(f"reverse_integer(-2147483648) = {reverse_integer(-2147483648)}") # Expected: 0 (overflow)

```

Up to Date

Integer reversal with overflow handling

Write a Python function to reverse an integer with 32-bit signed integer overflow handling.

Checking the current file for context.

Read task3.2.py

Improving overflow handling by checking during reversal instead of after.

Edit legal value for intellisense

Testing the current implementation:

Run command: cd python
cd "e:\user\ai-a-coding-v2\assessment2.5" & python task3.2.py

At C:\Users\user\AppData\Local\Temp\py-script-f1ea-4912-aa47-7d7a\code2118.ps
+ \$t = [System.Text.StringBuilder]::new()
+ \$t.Append("reverse_integer(123) = 321")
+ \$t.Append("reverse_integer(120) = 21")
+ \$t.Append("reverse_integer(5) = 5")
+ \$t.Append("reverse_integer(0) = 0")
+ \$t.Append("reverse_integer(154236468) = 0")
+ \$t.Append("reverse_integer(-2147483648) = 0")
Use Advisor? Success

Filing the PowerShell command:

Run command: cd python
cd "e:\user\ai-a-coding-v2\assessment2.5"; python task3.2.py

reverse_integer(123) = 321
reverse_integer(120) = 21
reverse_integer(5) = 5
reverse_integer(0) = 0
reverse_integer(154236468) = 0
reverse_integer(-2147483648) = 0

The function is implemented and working. It:
1. Reverses an integer by extracting digits and building the reversed number
2. Handles negative numbers by preserving the sign
3. Checks for 32-bit signed integer overflow — returns 0 if the reversed number is outside [-2,147,483,648, 2,147,483,647]

Test results:
• 123 → 321
• -123 → -321
• 120 → 21 (trailing zeros removed)
• 5 → 5
• 0 → 0
• 154236468 → 0 (overflow)
• -2147483648 → 0 (overflow)

The function correctly handles edge cases and overflow. The implementation checks bounds after reversal, which works since Python integers are unbounded and we only need to validate the final result against 32-bit limits.

Would you like me to optimize it to check for overflow during reversal, or is the current implementation sufficient?

Review

Plan, ⌂ for context, / for commands

Ctrl+K to generate command

Cursor Tab

```
task3.3.py - AI-A-coding-v2 - Cursor
```

```
task3.3.py - AI-A-coding-v2 - Cursor
```

```
task3.3.py - AI-A-coding-v2 - Cursor
```

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