

Assignment – 2.5

Name: CH.Harshini

Roll Number: 2303A51185

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 file 'task1.py' open. The code is divided into two sections: 'Original Code (Legacy Style)' and 'Improved/Refactored Code'. The terminal below shows the execution of the code and its output.

```
# 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 Code (Refactored)
def calculate_sums_improved(numbers):
    ...
    Calculate the sum of odd and even numbers in a list.

# 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 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}")

# 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}")
```

PS E:\Sem6\AI-A-coding-v2> & 'c:\Python314\python.exe' 'c:\Users\sprus\cursor\extensions\ms-python.debugger-2025.18.0-win32-x64\bundled\libs\debugger\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>

```

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    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_sum = sum(filter(lambda x: x % 2 != 0, numbers))
20     even_sum = sum(filter(lambda x: x % 2 == 0, numbers))
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 odd2, even2 = calculate_sums_improved(numbers)
26 print("Improved Code:")
27 print(f"Sum of odd numbers: {odd2}")
28 print(f"Sum of even numbers: {even2}")
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}")

```

```

task1.py (19)
# 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."""
    odd2 = sum(filter(lambda x: x % 2 != 0, numbers))
    even2 = sum(filter(lambda x: x % 2 == 0, numbers))
    return odd2, even2

# 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?

```

```

Python Debug Console
PS 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\sprus\cursor\extensions\ms-python-18.0-win32-x64\bundled\libs\debug\launcher' '61638' ... 'e:\sem6\AI-A-coding-v2\Assessment2.5\task1-2.py'
Improved Code:
Sum of odd numbers: 25
Sum of even numbers: 30
Alternative Improved Code:
Sum of odd numbers: 25
Sum of even numbers: 30
PS E:\sem6\AI-A-coding-v2>

```

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 assist session open. The main area displays a simple HTML file:

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Sample HTML Page</title>
7     <link rel="stylesheet" href="m.css">
8   </head>
9   <body>
10    <h1>welcome to SRU</h1>
11  </body>
12 </html>

```

Below the code editor, a "Prompts to try" section contains a button labeled "How do I use Gemini Code Assist?". A modal window titled "Ask Gemini or type '@'" is open, showing "1 context item" and buttons for "Default", "Agent", and "Preview".

The screenshot shows the AI-A-coding-v2 interface with a code assist session open. The main area displays a Python script for calculating the area of different shapes:

```

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)")

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

```

The interface provides an "Explanation" section with details about the function's behavior and logic. It includes sections for "Function Definition", "Conditional Logic", and "Example Usage". At the bottom, there is a "Ask Gemini or type '@'" modal with "1 context item" and "Agent", "Preview" buttons.

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

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, it lists project files including `task1.py`, `task1-2.py`, `task2.py`, `task3.py`, `Assignment-1.5`, `AI Assisted Coding-...`, `README.md`, `task2.py`, `task3.py`, `task5_iterative.py`, and `task5_recursive.py`.
- Code Editor:** The main area displays the `task3.py` file. The code is as follows:

```
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
20
```

- Terminal:** At the bottom, the terminal shows the command-line output of running the task3.py script.
- Python Debug Console:** A floating window titled "Python Debug Console" shows the execution of the script.
- Status Bar:** The status bar at the bottom right shows "21m" (21 minutes).

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

```

File Edit Selection View Go Run Terminal Help
task1.py task2.py task3.py task3.2.py task4.py task5.py
Assessment2.5
1 def reverse_integer(x):
2     """
3         Reverse an integer
4         Args:
5             x: An integer to reverse
6         Returns:
7             The reversed integer, or 0 if the reversed integer overflows 32-bit signed integer range
8             (i.e., outside [-2^31, 2^31 - 1] = [-2147483648, 2147483647])
9     """
10    # Handle negative numbers
11    is_negative = x < 0
12    num = abs(x)
13
14    # Reverse the digits
15    reversed_num = 0
16    while num > 0:
17        reversed_num = reversed_num * 10 + num % 10
18        num /= 10
19
20    # Apply sign
21    if is_negative:
22        reversed_num = -reversed_num
23
24    # Check for 32-bit signed integer overflow
25    INT_MIN = -2**31 # -2147483648
26    INT_MAX = 2**31 - 1 # 2147483647
27
28    if reversed_num < INT_MIN or reversed_num > INT_MAX:
29        return 0
30
31    return reversed_num
32
33
34
35
36
37
38
39
39 # __name__ == "__main__":
40 # Test case 1: Positive number
41 print("reverse_integer(123) = (reverse_integer(123))" # Expected: 321
42
43 # Test case 2: Negative number
44 print("reverse_integer(-123) = (reverse_integer(-123))" # Expected: -321
45
46 # Test case 3: Number ending with zero
47 print("reverse_integer(120) = (reverse_integer(120))" # Expected: 21
48
49 # Test case 4: Single digit
50 print("reverse_integer(5) = (reverse_integer(5))" # Expected: 5
51
52 # Test case 5: Zero
53 print("reverse_integer(0) = (reverse_integer(0))" # Expected: 0
54
55 # Test case 6: Overflow case (reversed number exceeds INT_MAX)
56 print("reverse_integer(1534236489) = (reverse_integer(1534236489))" # Expected: 0 (overflow)
57
58 # Test case 7: Overflow case (reversed number exceeds INT_MIN)
59 print("reverse_integer(-2147483648) = (reverse_integer(-2147483648))" # Expected: 0 (overflow)

```

Upgrade to Pro

Integer reversal with overflow handling

Writing 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 Illegal value for maxlen

Reading the file size?

Read task3.2.py

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

Edit Illegal value for maxlen

Reversing the file with improved overflow handling

Edit Illegal value for maxlen

Checking the file and trying a different approach

Read task3.2.py

Improving overflow handling by checking during reversal instead of after

Edit Illegal value for maxlen

Testing the current implementation:

Run command: cd python
cd "e:\sem2\AI-A-coding-v2\Assessment2.5" & python task3.2.py

Use AllowSet + Run

Problems Output Debug Console Terminal Ports
PowerShell Extension (2025.4.0)
Copyright (c) Microsoft Corporation.
https://www.microsoft.com/powershell
Type "help" to get help.

PS E:\sem2\AI-A-coding-v2

Cursor Tab

1 file
Plan, @ for context, / for commands
(cd Agent Auto -x
Cursor Tab
Un 56 Col 103 Spaces: 4 UTF-8 CRLF Python Python 3.14 (64-bit) Go Live

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

```

File Edit Selection View Go Run Terminal Help
task1.py task2.py task3.py task3.2.py task4.py task5.py
Assessment2.5
1 # Write a Python function to reverse an integer with 32-bit signed integer overflow handling.
2
3 def reverse_integer(x):
4     """
5         Reverse an integer with 32-bit signed integer overflow handling.
6         Args:
7             x: An integer to reverse
8         Returns:
9             The reversed integer, or 0 if the reversed integer overflows 32-bit signed integer range
10            (i.e., outside [-2^31, 2^31 - 1] = [-2147483648, 2147483647])
11
12    # Handle negative numbers
13    is_negative = x < 0
14    num = abs(x)
15
16    # Reverse the digits
17    reversed_num = 0
18    while num > 0:
19        reversed_num = reversed_num * 10 + num % 10
20        num /= 10
21
22    # Apply sign
23    if is_negative:
24        reversed_num = -reversed_num
25
26    # Check for 32-bit signed integer overflow
27    INT_MIN = -2**31 # -2147483648
28    INT_MAX = 2**31 - 1 # 2147483647
29
30    if reversed_num < INT_MIN or reversed_num > INT_MAX:
31        return 0
32
33    return reversed_num
34
35
36
37
38
39
39 # __name__ == "__main__":
40 # Test case 1: Positive number
41 print("reverse_integer(123) = (reverse_integer(123))" # Expected: 321
42
43 # Test case 2: Negative number
44 print("reverse_integer(-123) = (reverse_integer(-123))" # Expected: -321
45
46 # Test case 3: Number ending with zero
47 print("reverse_integer(120) = (reverse_integer(120))" # Expected: 21
48
49 # Test case 4: Single digit
50 print("reverse_integer(5) = (reverse_integer(5))" # Expected: 5
51
52 # Test case 5: Zero
53 print("reverse_integer(0) = (reverse_integer(0))" # Expected: 0
54
55 # Test case 6: Overflow case (reversed number exceeds INT_MAX)
56 print("reverse_integer(1534236489) = (reverse_integer(1534236489))" # Expected: 0 (overflow)
57
58 # Test case 7: Overflow case (reversed number exceeds INT_MIN)
59 print("reverse_integer(-2147483648) = (reverse_integer(-2147483648))" # Expected: 0 (overflow)

```

Upgrade to Pro

Integer reversal with overflow handling

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

Read task3.2.py

Improving overflow handling by checking during reversal instead of after

Edit Illegal value for maxlen

Testing the current implementation:

Ran command: cd python
cd "e:\sem2\AI-A-coding-v2\Assessment2.5" & python task3.2.py

At C:\Users\prince\AppData\Local\Temp\py-script-#f1c1bb-f70a-4912-aed7-7d7aaec82118.ps
1:7 char:43
+ cd "e:\sem2\AI-A-coding-v2\Assessment2.5" & python task3.2.py
+
The token 'cd' is not a valid statement separator in this version.
+ + CategoryInfo : ParserError: (:) [], ParentContainsErrorRecordException
+ FullyQualifiedErrorId : InvalidStatementEnd

Failing the PowerShell command:

Ran command: cd python
cd "e:\sem2\AI-A-coding-v2\Assessment2.5"; python task3.2.py

reverse_integer(123) = 321
reverse_integer(-123) = -321
reverse_integer(120) = 21
reverse_integer(5) = 5
reverse_integer(0) = 0
reverse_integer(1534236489) = 0
reverse_integer(-2147483648) = 0

Use AllowSet + Run

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 cases:
• 123 → 321
• -123 → -321
• 120 → 21 (trailing zeros removed)
• 5 → 5
• 1534236489 → 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?

1 file
Plan, @ for context, / for commands
(cd Agent Auto -x
Cursor Tab
Un 56 Col 103 Spaces: 4 UTF-8 CRLF Python Python 3.14 (64-bit) Go Live

The image shows two side-by-side screenshots of the AI-A Coding interface. Both screens display the same Python code for reversing an integer with strict overflow handling.

```

task3.3.py - AI-A-coding-v2 - Cursor
-----
# Write an optimized Python solution for reversing an integer with strict overflow checks for coding interviews.

def reverse_integer(x):
    """
    Optimized solution to reverse an integer with strict 32-bit signed integer overflow handling.

    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 0 if the reversed integer overflows 32-bit signed integer range
        (i.e., outside [-2^31, 2^31 - 1] = [-2147483648, 2147483647])

    Raises:
        ValueError: If result * 10 + digit > INT_MAX
        ValueError: If result < INT_MIN // 10 or (result == INT_MAX // 10 and digit > INT_MAX % 10)
        ValueError: If result < INT_MIN // 10 or (result == INT_MIN // 10 and digit < INT_MIN % 10)

    # Strict overflow check BEFORE multiplication
    # Check if result * 10 + digit would overflow
    if result > INT_MAX // 10 or (result == INT_MAX // 10 and digit > INT_MAX % 10):
        return 0

    # For positive overflow: result * 10 + digit > INT_MAX
    # Reversing result * (INT_MIN - digit) / 10
    if result < INT_MIN // 10 or (result == INT_MIN // 10 and digit < INT_MIN % 10):
        return 0

    # 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|)
    """

```

Top Screen (Left):

- File menu: File, Edit, Selection, View, Go, Run, Terminal, Help.
- Terminal tab: Shows command-line history including `cd ..` and `cd ./Assessment2.5`.
- Code editor: Displays the Python code for `reverse_integer` and `reverse_integer_v2` functions.
- Output tab: Shows PowerShell Extension output for task3.3.py.
- Bottom status bar: Shows file path (E:\sem\AI-A-coding-v2\), line count (21), column count (0), and file type (Python).

Bottom Screen (Right):

- File menu: Upgrade to Pro, File, Edit, Selection, View, Go, Run, Terminal, Help.
- Terminal tab: Shows command-line history including `cd ..` and `cd ./Assessment2.5`.
- Code editor: Displays the Python code for `reverse_integer` and `reverse_integer_v2` functions.
- Output tab: Shows the results of running `task3.3.py` and a performance comparison table.
- Bottom status bar: Shows file path (E:\sem\AI-A-coding-v2\), line count (19), column count (0), and file type (Python).

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