

Assignment – 2.5

Name: s.vignesh

Roll Number: 2303A51217

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 a project structure with files like README.md, task1.py, task2.py, task3.py, task5_iterative.py, and task5_recursive.py.
- Code Editor:** The active file is `task1.py`. It contains the following code: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}")
29
- Terminal:** Shows the command line output of the Python interpreter running the script.
- Output:** Shows the results of the original code execution.
- Python Debug Console:** Shows the command line output of the Python interpreter running the script.
- Status Bar:** Shows the current file is `task1.py`, the line is 28, column is 38, and the space count is 4.

```

task1-2.py - AI-A-coding-v2 - Cursor
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
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 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}")

Task1-2.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_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(f"\nImproved 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(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

The screenshot shows the AI-A-coding-v2 interface with a code editor and a preview channel. The code editor contains a Python script for calculating the area of different shapes. The preview channel shows the output of the script, which prints the area for a rectangle, circle, triangle, and square. A tooltip from GEMINI CODE ASSIST-HAT provides information about the calculate_area function, including its docstring and parameter descriptions. The bottom of the interface has a toolbar with buttons for Reject, Accept, and Edit.

```
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=7)
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}")

# Example of an error:
square_area = calculate_area("square", side=5)
print(f"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":
    o If the shape is "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 "circle", it checks for a radius.
    o If a radius is provided, 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 "triangle", it checks for a base and height.
    o If they are present, it calculates the area using the formula  $0.5 * \text{base} * \text{height}$ .
    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:
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.
```

The screenshot shows the AI-A-coding-v2 interface with a code editor and a preview channel. The code editor contains the same Python script as the previous screenshot. The preview channel shows the output of the script. A tooltip from GEMINI CODE ASSIST-HAT provides a detailed explanation of the calculate_area function, including its docstring, parameter descriptions, and logic for calculating the area of rectangles, circles, and triangles. The bottom of the interface has a toolbar with buttons for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, POSIX, and QUIT.

```
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=7)
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}")

# Example of an error:
square_area = calculate_area("square", side=5)
print(f"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":
    o If the shape is "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 "circle", it checks for a radius.
    o If a radius is provided, 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 "triangle", it checks for a base and height.
    o If they are present, it calculates the area using the formula  $0.5 * \text{base} * \text{height}$ .
    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:
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.
```

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

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

```

File Edit Selection View Go Run Terminal Help task1.py U task2.py U task3.py U task3.py X
github v Assessment2.5 > task3.py ...
Assessment2.5 > task3.py ...
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

```

Problems Output Debug Console Terminal Ports Upgrade to Pro ...

PS E:\sem6\AI-A-coding-v2> & 'c:\Python314\python.exe' 'c:\Users\sprous\cursor\extensions\ms-python-on-debug-2025.18.0-win32-x64\bin\debug\launcher' '59879' ... 'e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
PS E:\sem6\AI-A-coding-v2> 9547
PS E:\sem6\AI-A-coding-v2> & 'c:\Python314\python.exe' 'c:\Users\sprous\cursor\extensions\ms-python-on-debug-2025.18.0-win32-x64\bin\debug\launcher' '62775' ... 'e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
PS E:\sem6\AI-A-coding-v2> & 'c:\Python314\python.exe' 'c:\Users\sprous\cursor\extensions\ms-python-on-debug-2025.18.0-win32-x64\bin\debug\launcher' '62814' ... 'e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
PS E:\sem6\AI-A-coding-v2> 8520
PS E:\sem6\AI-A-coding-v2> 1

Task3.py lines 1-9

Cursor Tab Ln 20 Col 1 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 U task2.py U task3.py U task3.2.py U task3.2.py X
Assessment2.5 > task3.2.py ...
1 # reverse_integer(x):
2     """Reverse a negative numbers
3     if is_negative:
4         num = abs(x)
5
6     # Reverse the digits
7     reversed_num = 0
8     while num > 0:
9         reversed_num = reversed_num * 10 + num % 10
10        num /= 10
11
12    # Apply sign
13    if is_negative:
14        reversed_num = -reversed_num
15
16    # Check for 32-bit signed integer overflow
17    INT_MIN = -2**31 - 2147483648
18    INT_MAX = 2**31 - 1 - 2147483647
19
20    if reversed_num < INT_MIN or reversed_num > INT_MAX:
21        return 0
22
23    return reversed_num
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

```

Integer reversal with 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 btrehunder

Reading the file again

Read task3.2.py

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

Edit illegal value for btrehunder

Reverting the file with improved overflow handling.

Edit illegal value for btrehunder

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 btrehunder

Testing the current implementation:

Run command: cd

cd "e:\sem6\AI-A-coding-v2\Assessment2.5" & python task3.2.py

Use Allowed: ✓

2 of 2 Undo Ctrl+N Open Certificate

PowerShell Extension (2025.4.0)

Copyright (c) Microsoft Corporation.

<https://www.microsoft.com/powershell>

Type "help" to get help.

PS E:\sem6\AI-A-coding-v2>

Problems Output Debug Console Terminal Ports

Upgrading to Pro ...

... Agent Auto ...

New Chat

Plans, @ for context, / for commands

Local

Cursor Tab Ln 50 Col 103 Spaces: 4 UTF-8 CRLF Python Python 3.14 (64-bit) Go Live

The screenshot shows a Windows desktop environment with several windows open. The primary window is titled "task3.2.py - AI-A-coding-v2 - Cursor", displaying Python code for reversing integers. The code handles both positive and negative 32-bit signed integers, including edge cases like zero and negative numbers. It uses a recursive approach with base cases for 0 and 1-digit numbers. The code also includes comments explaining its logic and handling of integer overflow.

```
# Write a Python function to reverse an integer with 32-bit signed integer overflow handling.
# reverse_integer(0) == 0
# reverse_integer(1) == 1
# reverse an integer with 22-bit signed integer overflow handling.
# reverse_integer(-2**31) == -2**31
# reverse_integer(2**31 - 1) == 2**31 - 1
# reverse_integer(2**31) == 2**31 - 1
# reverse_integer(-2**31 - 1) == -2**31 - 1
# reverse_integer(2**31 + 1) == -2**31 - 1
# reverse_integer(-2**31 - 2) == 2**31 - 2
# reverse_integer(2**31 + 2) == -2**31 - 2
# reverse an integer with 32-bit signed integer overflow range
# i.e., outside [-2**31, 2**31 - 1] = [-2147483648, 2147483647]
# Handle negative numbers
# x is negative if < 0
num = abs(x)

# Reverse the digits
reversed_num = 0
while num != 0:
    reversed_num = reversed_num * 10 + num % 10
    num //= 10

# Edge case
if x < 0:
    reversed_num = -reversed_num

# Check for 32-bit signed integer overflow
INT_MIN = -2**31 # -2147483648
INT_MAX = 2**31 - 1 # 2147483647

if reversed_num < INT_MIN or reversed_num > INT_MAX:
    return 0

return reversed_num
```

The "Upgrade to Pro" window is visible at the top right. Below it, a window titled "Integer reversal with overflow handling" contains a command-line interface for testing the implementation. A PowerShell window titled "PowerShell Extension v2025.4.0" is also open. A terminal window at the bottom shows the command "PS E:\User\AI-A-coding-v2" and a help message from the "mscode-powershell" module.

The screenshot shows a code editor with several tabs open. The active tab is `task3.py - AI-A-coding-v2 - Cursor`. The code implements an optimized solution for reversing an integer with strict overflow checks. It includes comments explaining the complexity and space requirements, and a docstring for the `reverse_integer` function.

```
# Write an optimized Python solution for reversing an integer with strict overflow checks for coding interviews.
# Optimized solution to reverse an integer with strict 32-bit signed integer overflow handling.

Time Complexity: O(log(N)) - 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., [-231, 231 - 1] <= x <= 231 - 1) # 231 = 2147483648

    If result is 32-bit signed integer bounds
    INT_MIN = -231 = -2147483648
    INT_MAX = 231 - 1 = 2147483647

result = 0

while x != 0:
    # Extract the last digit
    digit = x % 10 if x > 0 else -(x // 10) # Handle negative modulo correctly
    x = x // 10 if x > 0 else -(x // 10) # Handle negative division correctly

    # Strict overflow check BEFORE multiplication
    # Check if result * 10 + digit would overflow
    # result * 10 + digit > INT_MAX

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

    # for negative overflow: result * 10 + digit < INT_MIN
    # for positive overflow: result * 10 + digit > INT_MAX
    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

Time Complexity: O(log(N))

# Alternative optimized version (more Pythonic and cleaner)
def reverse_integer_v2(x):
    ...
    Alternative optimized solution - cleaner approach handling

Time Complexity: O(log(N))
```

Below the code editor, there's a terminal window showing a PowerShell session with the command `cd "C:\sem\AI-A-coding-v2\Assessment2.5"; python task3.3.py`.

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