

Assignment – 2.5

Name: G. Rithika

Roll Number:2303A51190

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 with files like `task1.py`, `task2.py`, `task3.py`, `task5_iterative.py`, and `task5_recursive.py`.
- Editor:** Displays the code for `task1.py`. The code is annotated with AI-generated comments and scenarios. It defines a function `calculate_sums_original` to calculate the sum of odd and even numbers in a list.
- Terminal:** Shows the command-line output of running the script with the number list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. The output shows the original code's results: Sum of odd numbers: 25 and Sum of even numbers: 30.
- Status Bar:** Includes information like the file name (`task1.py - AI-A-coding-v2 - Cursor`), Python version (`Python 3.14 (64-bit)`), and other development tools.

```
# 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("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}")

# 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> ^C
PS E:\sem6\AI-A-coding-v2> e; cd 'e:\sem6\AI-A-coding-v2'; & 'c:\Python31A\python.exe' 'c:\Users\spursj.\cursor\extensions\vs-python-18.0-win32-x64\bundle\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

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

File Edit Selection View Go Run Terminal Help

GEMINI CODE ASSIST: CHAT

Preview Channel

Hello, Katakam
How can Gemini help?

◆ Gemini 3 Flash has been released
View our docs to learn more about how to start using Gemini 3 Flash in Code Assist.
[View docs](#)

Tips for getting started

- Open a file, get code suggestions as you type, and press **tab** to accept
- Press **ctrl+I** to ask Gemini to create or modify code
- Select code in the editor to reveal additional Gemini actions, such as **Explain Code**
- Pass in files to take advantage of Gemini Code Assist's generous

The set project ID (mega-alliance-jqmhf) was invalid, or the current account lacks permission to view it.

Source: Gemini Code Assist

Open Settings

LN 7, Col 39 Spaces: 4 UTF-8 CRLF { Python 3.13.5 (base) 11:53 ENG IN 16-01-2026

File Edit Selection View Go Run Terminal Help

GEMINI CODE ASSIST: CHAT

Preview Channel

```
print("The area of the rectangle is: {rectangle_area}")
circle_area = calculate_area("circle", radius=5)
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}")

square_area = calculate_area("square", side=5)
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 specifies the shape's name to calculate (e.g., `'rectangle'`, `'circle'`, etc.).
- `kwargs`: A keyword argument that allows Python Native functions 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 the shape is == "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.
- If the shape is == "circle":
 - o If the shape is == "circle", it checks for a `radius`.
 - o If found, it calculates the area using the formula πr^2 , importing the `math` module to get an accurate value of `(math.pi)`.
 - o If the `radius` is missing, it returns an error.
- If the 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.
- 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:

```
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}")

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLens

PS E:\Gemini\AI-A-coding-v2 & "c:\Python34\python.exe" "c:\Users\sprush\vscode\extensions\ms-python.debugger-2025.10.0-win32-x64\unzipped\libs\debug\launcher" "164922"
The area of the rectangle is: 50
The area of the circle is: 153.93804002589895
The area of the triangle is: 16.0
The area of the square is: Error: Unsupported shape.

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 VS Code interface with the following details:

- File Explorer:** Shows files in the "AI-A-CODING-V2" folder, including task1.py, task1-2.py, task2.py, task3.py, task5.py, Assignment-1.5, Assignment-2..., README.md, and day1.py.
- Code Editor:** The "task3.py" file is open, containing the following Python code:

```
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:** The "Python Debug Console" tab shows the following command-line session:

```
PS E:\sem6\AI-A-coding-v2> & "c:\Python314\python.exe" "e:\Users\sprous\cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundles\libs\debugpy\launcher" "59870" -- "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> 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\bundles\libs\debugpy\launcher" "53775" -- "e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py"
PS 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\bundles\libs\debugpy\launcher" "52814" -- "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>
```
- Status Bar:** Shows "Cursor Tab" and other status indicators.

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

    # 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
    print(f"reverse_integer(-123) = {reverse_integer(-123)}") # Expected: -321

    # Test case 2: Negative number
    print(f"reverse_integer(123) + (reverse_integer(-123)) = {reverse_integer(123) + reverse_integer(-123)}") # Expected: 0

    # Test case 3: Number ending with zero
    print(f"reverse_integer(120) + (reverse_integer(120)) = {reverse_integer(120) + reverse_integer(120)}") # Expected: 24

    # 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(1534236489) = {reverse_integer(1534236489)}") # Expected: 0 (overflow)

    # Test case 7: Overflow case (reversed number exceeds INT_MIN)
    print(f"reverse_integer(-2147483648) + (reverse_integer(-2147483648)) = {reverse_integer(-2147483648) + reverse_integer(-2147483648)}") # Expected: 0 (overflow)

```

Upgrading to Pro ... ↗ ⓘ

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 `isnegative` value for `intnumber`

Reading the file again.

Read task3.2.py

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

Edit `isnegative` value for `intnumber`

Rewriting the file with improved overflow handling.

Edit `isnegative` value for `intnumber`

Checking the file and trying a different approach.

Read task3.2.py

Improving overflow handling by checking during reversal instead of after.

Edit `isnegative` value for `intnumber`

Testing the current implementation:

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

Skip Allow Edit Run

Use AllowEdit

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

OUTPUT TIMELINE

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

Upgrade to Pro ... ↗ ⓘ

Integer reversal with overflow handling

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

Reading the file again.

Read task3.2.py

Improving overflow handling by checking during reversal instead of after.

Edit `isnegative` value for `intnumber`

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-f1e0-4912-4a47-7d7a\code2116.ps
+ \$ = \$ + 1
+ cd "e:\user\ai-a-coding-v2\assessment2.5" & python task3.2.py
+
The token '\$=' is not a valid statement separator in this version.
+ CategoryInfo : ParserError: (:) [], ParentContainsErrorRecordException
+ FullyQualifiedErrorId : InvalidEndOfFileLine

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

Use AllowEdit ✓ Success

The function is implemented and working:
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
• 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?

Review

Plan, ⌂ for context, / for commands

Ctrl+K to generate command

Cursor Tab

The screenshot shows a terminal window titled "task3.py - AI-A-coding-v2 - Cursor" with the following content:

```
File Edit Selection View Go Run Terminal Help task3.py - AI-A-coding-v2 - Cursor Upgrade to Pro ... + ... Integer reversal with overflow handling

# Write an optimized Python solution for reversing an integer with strict overflow checks for coding interviews.

Assessment2.5 * task3.py
def reverse_Integer():
    """
    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., outside [-2^31, 2^31 - 1] = [-2147483648, 2147483647])

    # 32-bit signed integer bounds
    INT_MIN = -2**31
    INT_MAX = 2**31 - 1

    result = 0

    while x != 0:
        digit = x % 10
        x = x // 10

        # Strict overflow check BEFORE multiplication
        # Checks if result * 10 + digit would overflow
        if result * 10 + digit > INT_MAX:
            result = 0
            break

        # For positive overflow: result * 10 + digit > INT_MAX
        # If reversed result < INT_MIN, result = 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_2(x):
    """
    Alternative optimized solution - cleaner approach handling sign separately.
    """
    result = 0
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31

    for digit in str(x)[::-1]:
        result = result * 10 + int(digit)

        # If result < INT_MIN // 10 or (result == INT_MIN // 10 and digit < INT_MIN % 10):
        #     return 0

    return result

# Time Complexity: O(log(n))

PowerShell Extension v2025.4.0
Copyright (C) Microsoft Corporation.

https://aka.ms/vscode-powershell
Type 'help' to get help.

PS E:\vscode\AI-A-coding-v2\】
```

The terminal also displays the output of running the script:

```
Waiting for Approval
Result: Succeeded, follow up?
00 Agent Auto -
```

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