

# AI ASSISTANT CODING

## Assignment – 2.5

Roll Number: 2303A510F7

Batch – 03

### 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 PyCharm IDE interface with the following details:

- File Structure:** Shows a project named "AI-A-CODING-V2" containing files like .github, .vscode, Assessment2.5, task1.py, Assignment-1.5, README.md, task2.py, task3.py, task3\_iterative.py, and task5\_recursive.py.
- Code Editor:** The main editor window displays the original code for calculating odd and even sums from a list. The code uses a while loop and if-else statements to iterate through the list and update odd and even sums respectively.
- Review Panel:** A right-hand panel titled "task1.py lines 1-9" contains the code and instructions:
  - # 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
- Refactor Results:** Below the review panel, two sections show the "Original Code (Legacy Style)" and the "Improved/Refactored Code". The improved code is a more concise one-liner using list comprehension and the sum function.
- Terminal:** The bottom terminal window shows the command-line output of running the script with sample data [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

```

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
15    return odd_sum, even_sum
16
17 # Alternative Improved version using filter()
18 def calculate_sums_alternative(numbers):
19     """Alternative refactored version using filter."""
20     odd_sum = sum(filter(lambda x: x % 2 != 0, numbers))
21     even_sum = sum(filter(lambda x: x % 2 == 0, numbers))
22
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(f"\nImproved 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(f"\nAlternative Improved Code:")
35 print(f"Sum of odd numbers: {odd2}")
36 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_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}")

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

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. For example, if "rectangle".
    • kwargs: This is a standard 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 the shape is "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.
- If the shape is "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 `radius` is missing, it returns an error.
- If the shape is "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.
- 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.

Generated by Gemini-gemini-23-pro

Ask Gemini or type 'g'

1 context item Agent Preview

PS E:\vscode\AI-A-coding-v2 & "c:\Python39\python.exe" "c:\Users\sprus\vscode\extensions\es-py\python-debug-2025.10.0-win32-x64\undistilled\bin\debug\launcher" "64362" -l

PS E:\vscode\AI-A-coding-v2\Assessment2\5Task1.py

The area of the rectangle is: 50

The area of the circle is: 78.54000025898985

The area of the triangle is: 16.0

The area of the square is: Error: Unsupported shape.

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

## 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.

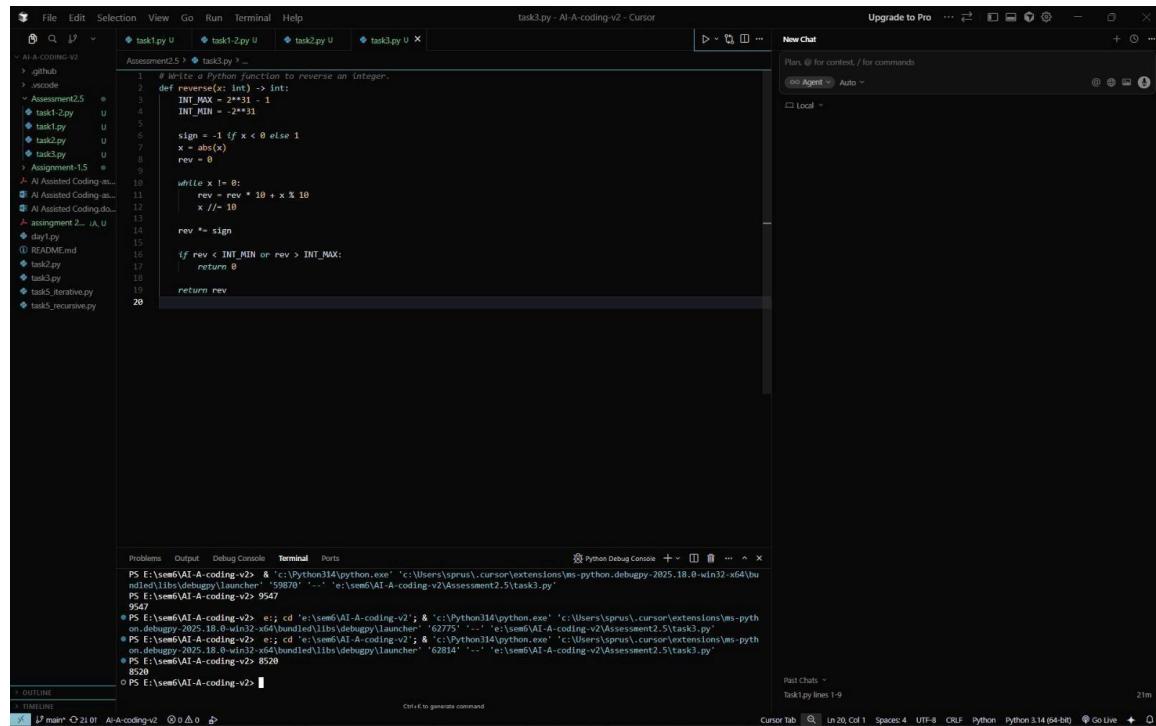
### ❖ Task:

Use Cursor AI with different prompts for the same problem and observe code changes.

### ❖ Expected Output:

#### ➤ Prompt list

#### ➤ Code variations



The screenshot shows a code editor interface with several tabs open. The active tab is 'task3.py - AI-A-coding-v2 - Cursor'. The code is a Python function to reverse an integer, with comments explaining the logic. The code is identical across all tabs shown. The tabs include 'task1.py U', 'task1-2.py U', 'task2.py U', and 'task3.py U'. The sidebar on the left shows a file tree with various files and folders, mostly related to 'AI Assisted Coding' and 'Assignment'. The bottom of the screen shows a terminal window with a command-line history. The terminal output includes several PS commands, indicating multiple runs of the AI tool. The terminal also shows some error messages and command-line arguments like '-l' and '-o'.

```
# Write a python function to reverse an integer.
def reverse(x: int) -> int:
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31
    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
```

```
PS E:\sem6\AI-A-coding-v2 & 'c:\Python314\python.exe' 'c:\Users\sprous1.cursor\extensions\ms-python.debugger-2025.18.0-win32-x64\bin\debug\launcher' '59870' -- -e '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> cd 'E:\sem6\AI-A-coding-v2'; & 'c:\Python314\python.exe' 'c:\Users\sprous1.cursor\extensions\ms-python.debugger-2025.18.0-win32-x64\bin\debug\launcher' '62775' -- -e 'E:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
PS E:\sem6\AI-A-coding-v2> cd 'E:\sem6\AI-A-coding-v2'; & 'c:\Python314\python.exe' 'c:\Users\sprous1.cursor\extensions\ms-python.debugger-2025.18.0-win32-x64\bin\debug\launcher' '62814' -- -e '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
```

File Edit Selection View Go Run Terminal Help task3.2.py - AI-A-coding-v2 - Cursor

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

def reverse_integer(x):
    """
    Reverse an integer with 32-bit signed integer overflow handling.

    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])
    """
    # Handle negative numbers
    is_negative = x < 0
    num = abs(x)

    # Reverse the digits
    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 # -2147483648
    INT_MAX = 2**31 - 1 # 2147483647
    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("reverse_integer(123) = (reverse_integer(123))" # Expected: 321
    # Test case 2: Negative number
    print("reverse_integer(-123) = (reverse_integer(-123))" # Expected: -321
    # Test case 3: Number ending with zero
    print("reverse_integer(120) = (reverse_integer(120))" # Expected: 21
    # Test case 4: Single digit
    print("reverse_integer(5) = (reverse_integer(5))" # Expected: 5
    # Test case 5: Zero
    print("reverse_integer(0) = (reverse_integer(0))" # Expected: 0
    # Test case 6: Overflow case (reversed number exceeds INT_MAX)
    print("reverse_integer(1534236468) = (reverse_integer(1534236468))" # Expected: 0 (overflows)
    # Test case 7: Overflow case (reversed number exceeds INT_MIN)
    print("reverse_integer(-2147483648) = (reverse_integer(-2147483648))" # Expected: 0 (overflows)
```

Upgrade 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 Illegal value for fileNumber

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 fileNumber

Rewriting the file with improved overflow handling.

Edit Illegal value for fileNumber

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 fileNumber

Testing the current implementations:

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

Use Allowlist Skip Allowlist for 1st Run

Problems Output Debug Console Terminal Ports

PowerShell Extension v0.25.4.0 Copyright (c) Microsoft Corporation. https://www.microsoft.com/powershell Type 'help' to get help.

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

OUTLINE TIMELINE

File Edit Selection View Go Run Terminal Help task3.2.py - AI-A-coding-v2 - Cursor

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

def reverse_integer(x):
    """
    Reverse an integer with 32-bit signed integer overflow handling.

    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])
    """
    # Handle negative numbers
    is_negative = x < 0
    num = abs(x)

    # Reverse the digits
    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 # -2147483648
    INT_MAX = 2**31 - 1 # 2147483647
    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("reverse_integer(123) = (reverse_integer(123))" # Expected: 321
    # Test case 2: Negative number
    print("reverse_integer(-123) = (reverse_integer(-123))" # Expected: -321
    # Test case 3: Number ending with zero
    print("reverse_integer(120) = (reverse_integer(120))" # Expected: 21
    # Test case 4: Single digit
    print("reverse_integer(5) = (reverse_integer(5))" # Expected: 5
    # Test case 5: Zero
    print("reverse_integer(0) = (reverse_integer(0))" # Expected: 0
    # Test case 6: Overflow case (reversed number exceeds INT_MAX)
    print("reverse_integer(1534236468) = (reverse_integer(1534236468))" # Expected: 0 (overflows)
    # Test case 7: Overflow case (reversed number exceeds INT_MIN)
    print("reverse_integer(-2147483648) = (reverse_integer(-2147483648))" # Expected: 0 (overflows)
```

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 Illegal value for fileNumber

Testing the current implementations:

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

At C:\Users\open\AppData\Local\Temp\ps-script-#f1e016-#0d-4012-aed7-7d7aaec0211a.ps
Line 17: + cd "e:\vscode\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 : InvalidToken

Fixing the PowerShell command:

Run command: cd python
cd "e:\vscode\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(1534236468) = 0
reverse\_integer(-2147483648) = 0

✓ 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
• 1534236468 → 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?

OUTLINE TIMELINE

File Edit Selection View Go Run Terminal Help task3.2.py - AI-A-coding-v2 - Cursor

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

def reverse_integer(x):
    """
    Reverse an integer with 32-bit signed integer overflow handling.

    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])
    """
    # Handle negative numbers
    is_negative = x < 0
    num = abs(x)

    # Reverse the digits
    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 # -2147483648
    INT_MAX = 2**31 - 1 # 2147483647
    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("reverse_integer(123) = (reverse_integer(123))" # Expected: 321
    # Test case 2: Negative number
    print("reverse_integer(-123) = (reverse_integer(-123))" # Expected: -321
    # Test case 3: Number ending with zero
    print("reverse_integer(120) = (reverse_integer(120))" # Expected: 21
    # Test case 4: Single digit
    print("reverse_integer(5) = (reverse_integer(5))" # Expected: 5
    # Test case 5: Zero
    print("reverse_integer(0) = (reverse_integer(0))" # Expected: 0
    # Test case 6: Overflow case (reversed number exceeds INT_MAX)
    print("reverse_integer(1534236468) = (reverse_integer(1534236468))" # Expected: 0 (overflows)
    # Test case 7: Overflow case (reversed number exceeds INT_MIN)
    print("reverse_integer(-2147483648) = (reverse_integer(-2147483648))" # Expected: 0 (overflows)
```

Review

Plan ⌂ for context, / for commands

Agent Auto ~ 1x

Cursor Tab Ln 98 Col 103 Space: 4 UTF-8 CRLF Python Python 3.14 (64-bit) Go Live

The screenshot displays a developer's environment with two terminal windows side-by-side, both running Python 3.8. The left terminal, titled "task3.3.py - AI-A-coding-v2 - Cursor", contains the original Python code for reversing an integer with strict overflow checks. The right terminal, titled "task3.3\_optimized.py - AI-A-coding-v2 - Cursor", contains the optimized version of the same code. Both terminals show the code executing successfully.

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

```
# 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., outside [-2^31, 2^31 - 1] = [-2147483648, 2147483647])

# 32-bit signed integer bounds
INT_MIN = -2**31
INT_MAX = 2**31 - 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
    if result < INT_MIN // 10 or (result == INT_MAX // 10 and digit > INT_MAX % 10):
        return 0

    # For positive overflow: result * 10 + digit > INT_MAX
    # For negative overflow: result * 10 + digit < INT_MIN
    # Round-trip: result < INT_MIN // 10 or (result == INT_MAX // 10 and digit < INT_MIN % 10)
    if result < INT_MIN // 10 or (result == INT_MAX // 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
```

**task3.3\_optimized.py - AI-A-coding-v2 - Cursor**

```
# 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., outside [-2^31, 2^31 - 1] = [-2147483648, 2147483647])

# 32-bit signed integer bounds
INT_MIN = -2**31
INT_MAX = 2**31 - 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
    if result < INT_MIN // 10 or (result == INT_MAX // 10 and digit > INT_MAX % 10):
        return 0

    # For positive overflow: result * 10 + digit > INT_MAX
    # For negative overflow: result * 10 + digit < INT_MIN
    # Round-trip: result < INT_MIN // 10 or (result == INT_MAX // 10 and digit < INT_MIN % 10)
    if result < INT_MIN // 10 or (result == INT_MAX // 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.
```

## 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