Assignment – 2.5

Name: N. Paul Benjamin

Roll Number: 2303A51116

Batch - 03

AI Assisted Coding

16-01-2026

Task 1: Refactoring Odd/Even Logic (List Version)

❖ Scenario:

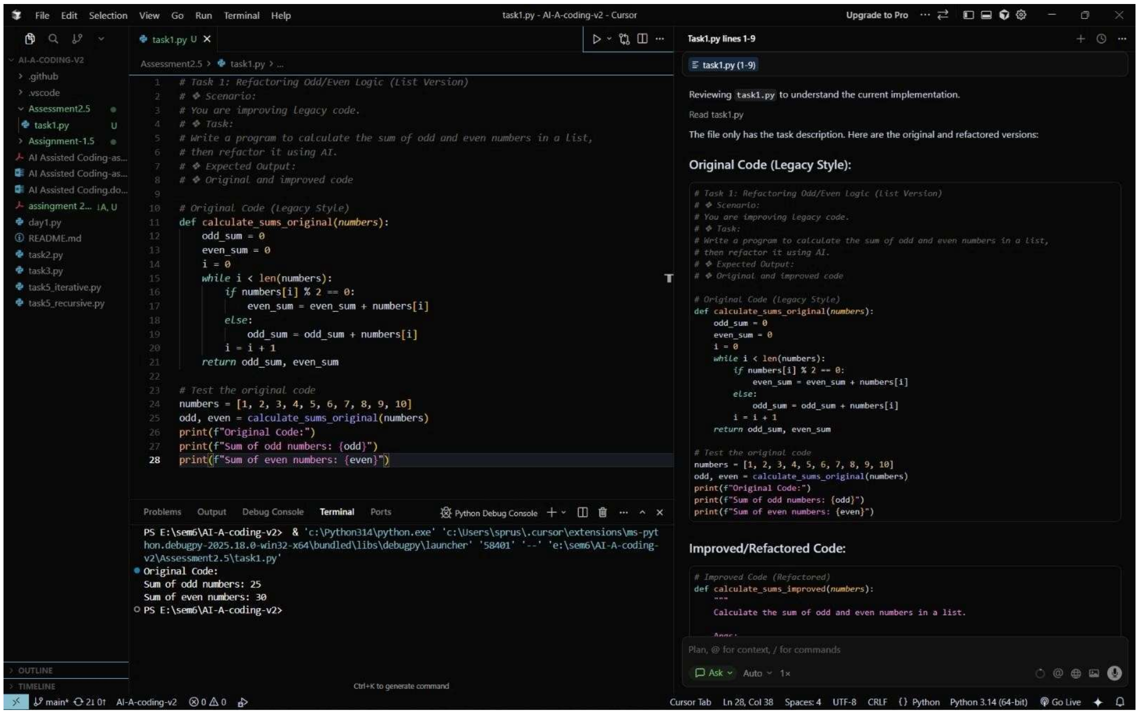
You are improving legacy code.

❖ 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


File Edit Selection View Go Run Terminal Help
task1.py - AI-A-coding-v2 - Cursor
Upgrade to Pro
task1.py U
AI-A-CODING-V2
.github
.vscode
Assessment2.5
task1.py
Assignment-1.5
AI Assisted Coding-as...
AI Assisted Coding-as...
AI Assisted Coding.do...
assingment 2... !A, U
day1.py
README.md
task2.py
task3.py
task5_iterative.py
task5_recursive.py
Assessment2.5 > task1.py > ...
OUTLINE
TIMELINE
Problems Output Debug Console Terminal Ports
Python Debug Console
Ctrl+K to generate command
main* 21 0↑ AI-A-coding-v2 0 0
Task1.py lines 1-9
task1.py (1-9)
Plan, @ for context, / for commands
Ask Auto 1×
Cursor Tab Ln 28, Col 38 Spaces: 4 UTF-8 CRLF {} Python Python 3.14 (64-bit) Go Live

```python
# 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(f"Original 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-pyt
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\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>
```

Reviewing `task1.py` to understand the current implementation.

Read task1.py

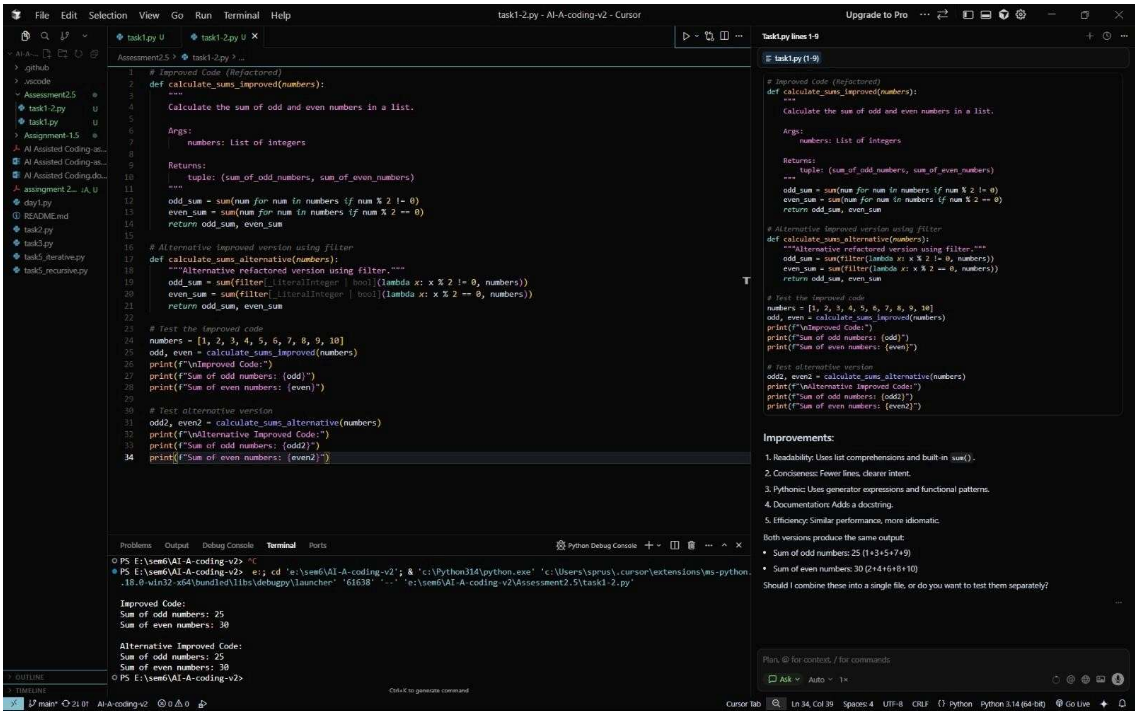
The file only has the task description. Here are the original and refactored versions:

Original Code (Legacy Style):

```python
# 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(f"Original Code:")
print(f"Sum of odd numbers: {odd}")
print(f"Sum of even numbers: {even}")
```

Improved/Refactored Code:

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



```python
# 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[_LiteralInteger | bool](lambda x: x % 2 != 0, numbers))
    even_sum = sum(filter[_LiteralInteger | bool](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}")
```

```
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\debugpy\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>
```

Task1.py lines 1-9

task1.py (1-9)

```python
# 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.

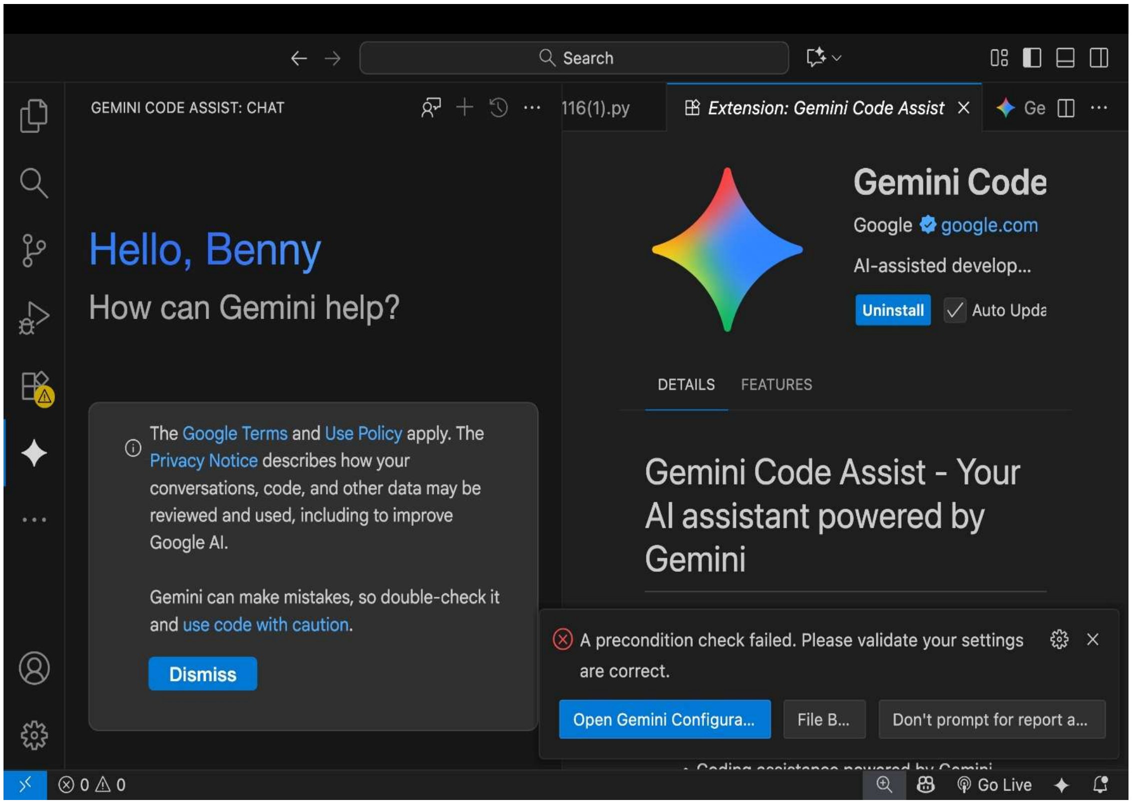
❖ Task:

Ask Gemini to explain a function that calculates the area of different shapes.

❖ Expected Output:

➢ Code

➢ Explanation



Hello, Benny

How can Gemini help?

The Google Terms and Use Policy apply. The Privacy Notice describes how your conversations, code, and other data may be reviewed and used, including to improve Google AI.

Gemini can make mistakes, so double-check it and use code with caution.

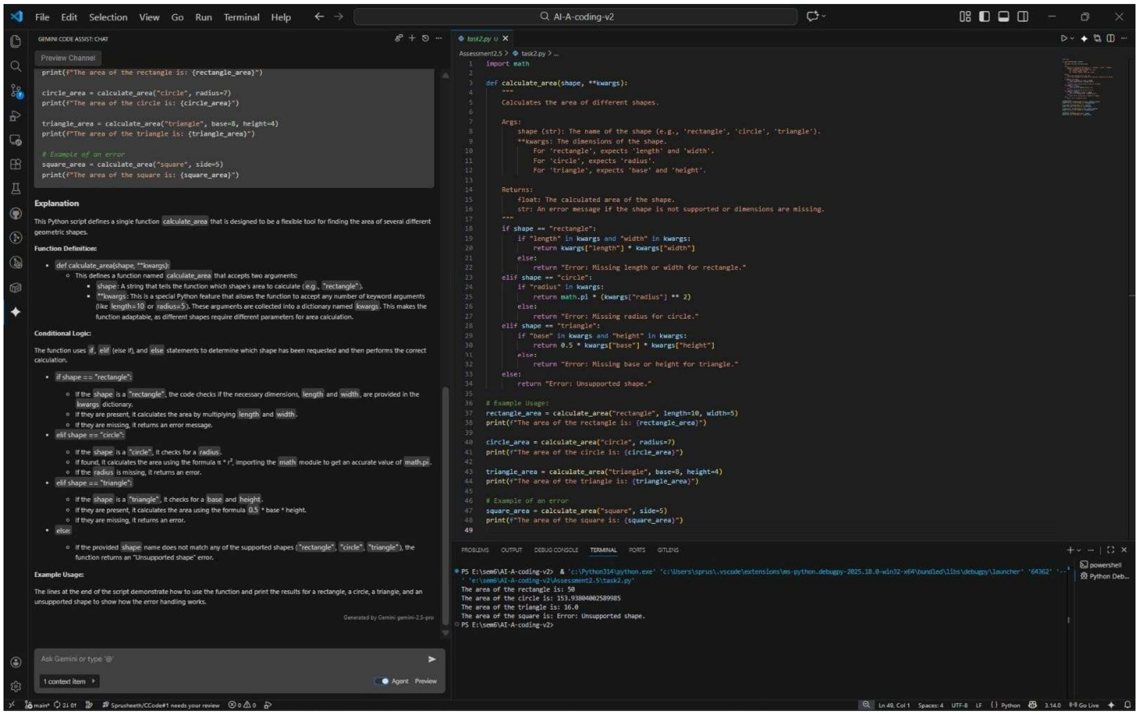
Gemini Code

Google google.com

AI-assisted develop...

Gemini Code Assist - Your AI assistant powered by Gemini

A precondition check failed. Please validate your settings are correct.



```
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}")
```

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 π * r², 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.

Generated by Gemini gemini-2.5-pro

```python
import math

def calculate_area(shape, **kwargs):
    """
    Calculates the area of different shapes.

    Args:
        shape (str): The name of the shape (e.g., 'rectangle', 'circle', 'triangle').
        **kwargs: The dimensions of the shape.
            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}")
```

```
PS E:\sem6\AI-A-coding-v2>  & 'c:\Python314\python.exe' 'c:\Users\sprus\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64362' '--'
 'e:\sem6\AI-A-coding-v2\Assessment2.5\task2.py'
The area of the rectangle is: 50
The area of the circle is: 153.93804002589985
The area of the triangle is: 16.0
The area of the square is: Error: Unsupported shape.
PS E:\sem6\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.

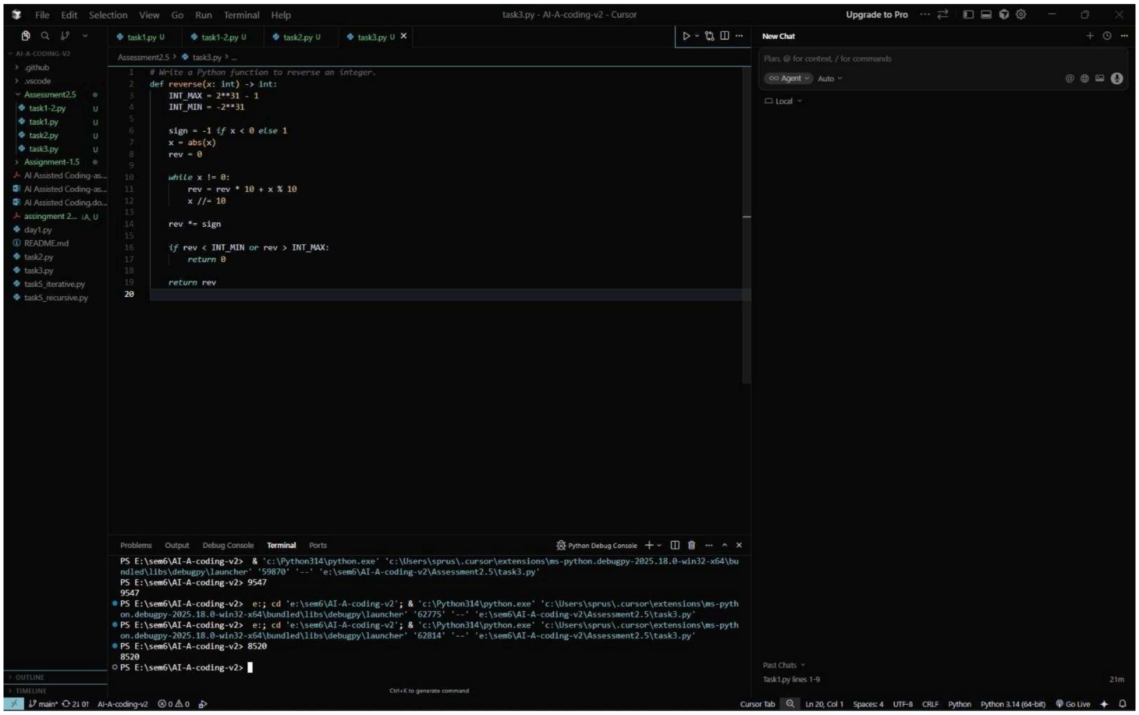
❖ Task:

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

❖ Expected Output:

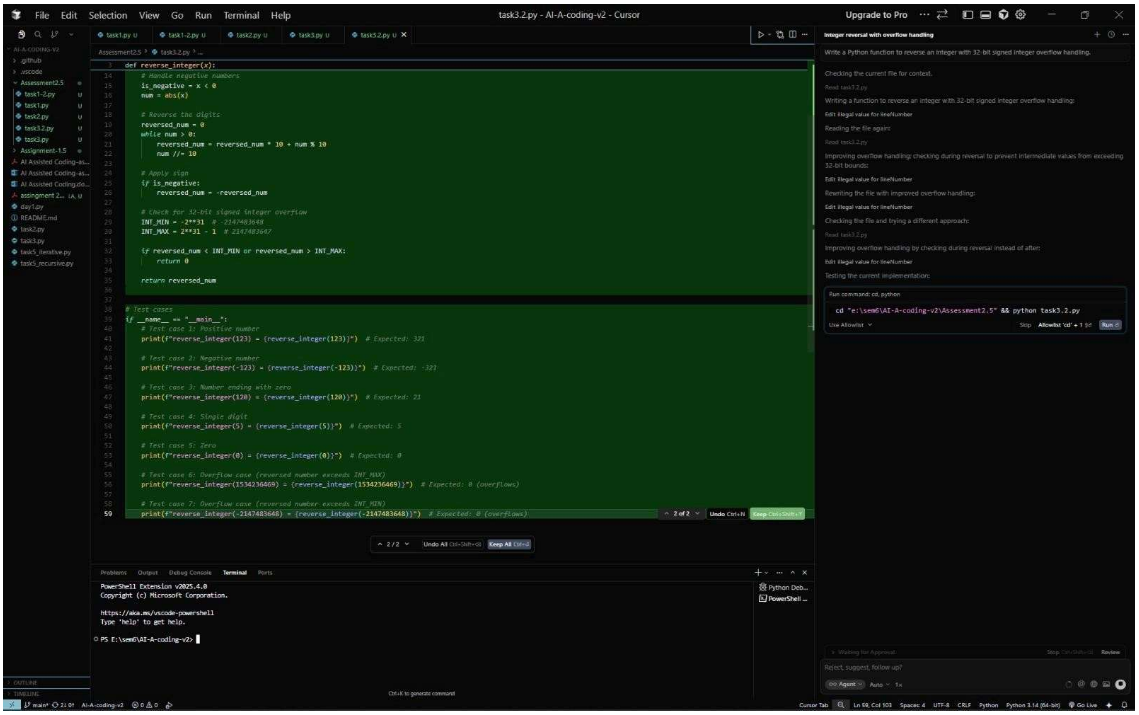
➢ Prompt list

➢ Code variations



```python
# 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\sprus\.cursor\extensions\ms-python.debugpy-2025.18.0-win32-x64\bu
ndled\libs\debugpy\launcher' '59870' '--' 'e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
PS E:\sem6\AI-A-coding-v2> 9547
9547
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-pyth
on.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62775' '--' '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\sprus\.cursor\extensions\ms-pyth
on.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62814' '--' 'e:\sem6\AI-A-coding-v2\Assessment2.5\task3.py'
PS E:\sem6\AI-A-coding-v2> 8520
8520
PS E:\sem6\AI-A-coding-v2> 
```



```python
def reverse_integer(x):
    # 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(f"reverse_integer(123) = {reverse_integer(123)}")  # Expected: 321

    # Test case 2: Negative number
    print(f"reverse_integer(-123) = {reverse_integer(-123)}")  # Expected: -321

    # Test case 3: Number ending with zero
    print(f"reverse_integer(120) = {reverse_integer(120)}")  # Expected: 21

    # Test case 4: Single digit
    print(f"reverse_integer(5) = {reverse_integer(5)}")  # Expected: 5

    # Test case 5: Zero
    print(f"reverse_integer(0) = {reverse_integer(0)}")  # Expected: 0

    # Test case 6: Overflow case (reversed number exceeds INT_MAX)
    print(f"reverse_integer(1534236469) = {reverse_integer(1534236469)}")  # Expected: 0 (overflows)

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

```
PowerShell Extension v2025.4.0
Copyright (c) Microsoft Corporation.

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

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

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 lineNumber

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 lineNumber

Rewriting the file with improved overflow handling:

Edit illegal value for lineNumber

Checking the file and trying a different approach:

Read task3.2.py

Improving overflow handling by checking during reversal instead of after:

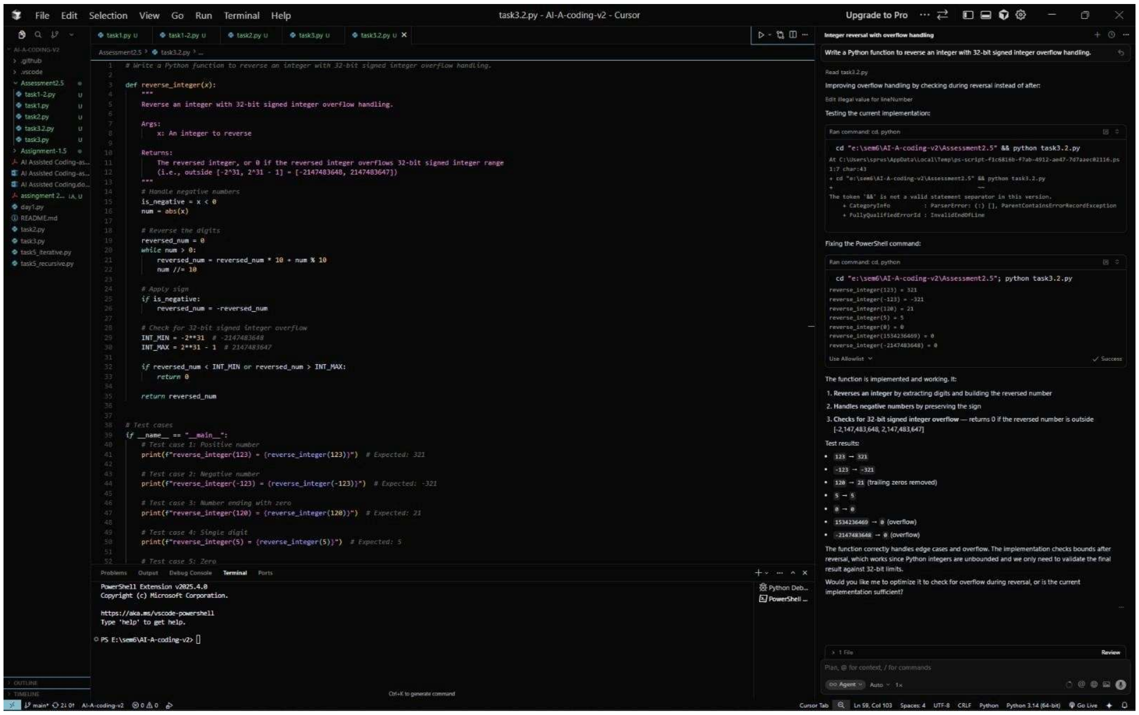
Edit illegal value for lineNumber

Testing the current implementation:

Run command: cd, python

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

Use Allowlist ⌄ Skip Allowlist 'cd' + 1 Run


File Edit Selection View Go Run Terminal Help
task3.2.py - AI-A-coding-v2 - Cursor
Upgrade to Pro
task1.py U
task1-2.py U
task2.py U
task3.py U
task3.2.py U
Assessment2.5 › task3.2.py › ...
Problems Output Debug Console Terminal Ports
Python Deb...
PowerShell ...
Ctrl+K to generate command
main* 2↓ 0↑ AI-A-coding-v2 ⊗ 0 ⚠ 0
Cursor Tab Ln 59, Col 103 Spaces: 4 UTF-8 CRLF Python Python 3.14 (64-bit) Go Live
1 File
Review
Plan, @ for context, / for commands
Agent Auto 1x

```python
# 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(f"reverse_integer(123) = {reverse_integer(123)}")  # Expected: 321

    # Test case 2: Negative number
    print(f"reverse_integer(-123) = {reverse_integer(-123)}")  # Expected: -321

    # Test case 3: Number ending with zero
    print(f"reverse_integer(120) = {reverse_integer(120)}")  # Expected: 21

    # Test case 4: Single digit
    print(f"reverse_integer(5) = {reverse_integer(5)}")  # Expected: 5

    # Test case 5: Zero
```

```
PowerShell Extension v2025.4.0
Copyright (c) Microsoft Corporation.

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

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

Integer reversal with overflow handling

Write 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 lineNumber

Testing the current implementation:

Ran command: cd, python

```
cd "e:\sem6\AI-A-coding-v2\Assessment2.5" && python task3.2.py
At C:\Users\sprus\AppData\Local\Temp\ps-script-f1c6818b-f7ab-4912-ae47-7d7aaec02116.ps
1:7 char:43
+ cd "e:\sem6\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 : InvalidEndOfLine
```

Fixing the PowerShell command:

Ran command: cd, python

```
cd "e:\sem6\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(1534236469) = 0
reverse_integer(-2147483648) = 0
```

Use Allowlist ⌄ ✓ Success

The function is implemented and working. It:

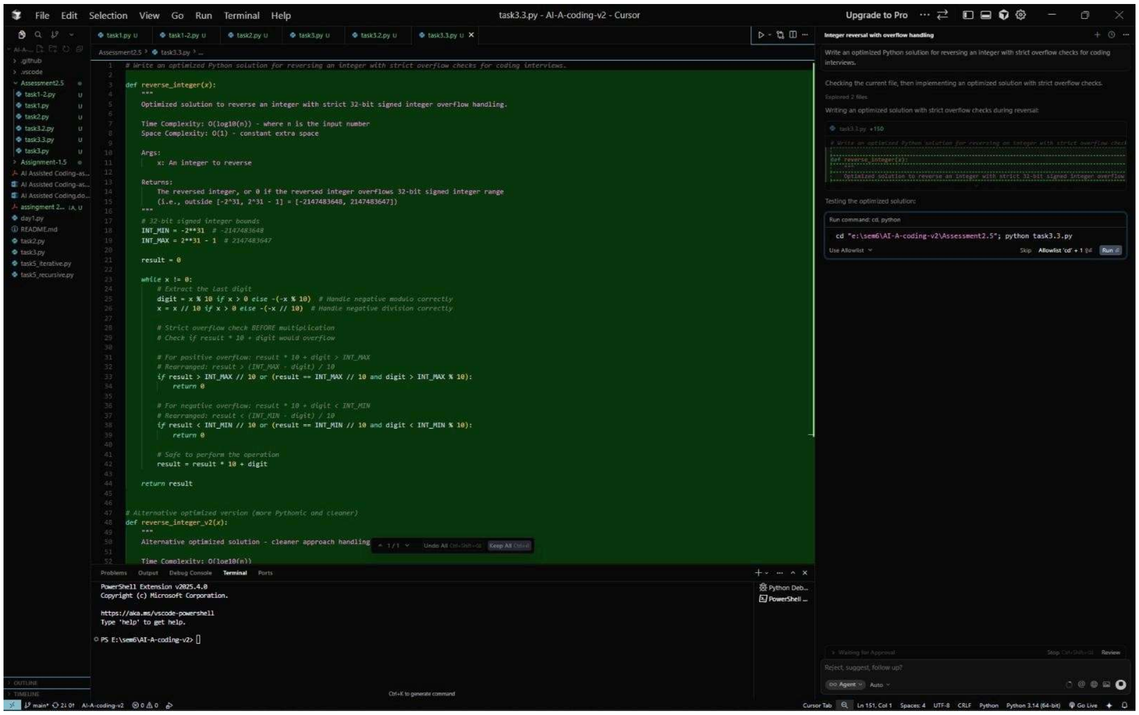
1. **Reverses an integer** by extracting digits and building the reversed number
2. **Handles negative numbers** by preserving the sign
3. **Checks for 32-bit signed integer overflow** — returns 0 if the reversed number is outside [-2,147,483,648, 2,147,483,647]

Test results:

- `123` → `321`
- `-123` → `-321`
- `120` → `21` (trailing zeros removed)
- `5` → `5`
- `0` → `0`
- `1534236469` → `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?



```python
# 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(log10(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  # -2147483648
    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

        # For positive overflow: result * 10 + digit > INT_MAX
        # Rearranged: result > (INT_MAX - digit) / 10
        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
        # Rearranged: 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

    Time Complexity: O(log10(n))
```

```
PowerShell Extension v2025.4.0
Copyright (c) Microsoft Corporation.

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

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

Integer reversal with overflow handling

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

Checking the current file, then implementing an optimized solution with strict overflow checks.

Explored 2 files

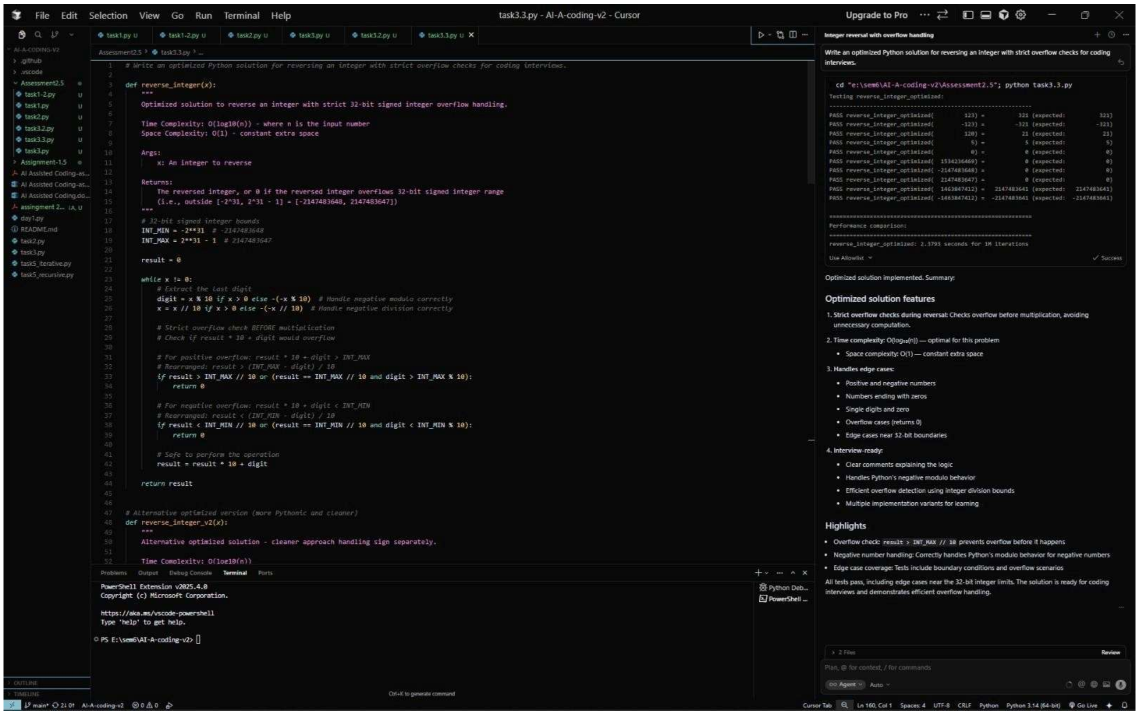
Writing an optimized solution with strict overflow checks during reversal:

task3.3.py +150

Testing the optimized solution:

Run command: cd, python

```
cd "e:\sem6\AI-A-coding-v2\Assessment2.5"; python task3.3.py
```



```python
# 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(log10(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  # -2147483648
    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

        # For positive overflow: result * 10 + digit > INT_MAX
        # Rearranged: result > (INT_MAX - digit) / 10
        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
        # Rearranged: 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(log10(n))
```

```
PowerShell Extension v2025.4.0
Copyright (c) Microsoft Corporation.

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

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

Integer reversal with overflow handling

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

```
cd "e:\sem6\AI-A-coding-v2\Assessment2.5"; python task3.3.py
Testing reverse_integer_optimized:
------------------------------------------------------------
PASS reverse_integer_optimized(        123) =        321 (expected:        321)
PASS reverse_integer_optimized(       -123) =       -321 (expected:       -321)
PASS reverse_integer_optimized(        120) =         21 (expected:         21)
PASS reverse_integer_optimized(          5) =          5 (expected:          5)
PASS reverse_integer_optimized(          0) =          0 (expected:          0)
PASS reverse_integer_optimized( 1534236469) =          0 (expected:          0)
PASS reverse_integer_optimized(-2147483648) =          0 (expected:          0)
PASS reverse_integer_optimized( 2147483647) =          0 (expected:          0)
PASS reverse_integer_optimized( 1463847412) =  2147483641 (expected:  2147483641)
PASS reverse_integer_optimized(-1463847412) = -2147483641 (expected: -2147483641)

============================================================
Performance comparison:
============================================================
reverse_integer_optimized: 2.3793 seconds for 1M iterations
```

Optimized solution implemented. Summary:

Optimized solution features

1. **Strict overflow checks during reversal:** Checks overflow before multiplication, avoiding unnecessary computation.
2. **Time complexity:** $O(\log_{10}(n))$ — optimal for this problem
   - Space complexity: O(1) — constant extra space
3. **Handles edge cases:**
   - Positive and negative numbers
   - Numbers ending with zeros
   - Single digits and zero
   - Overflow cases (returns 0)
   - Edge cases near 32-bit boundaries
4. **Interview-ready:**
   - Clear comments explaining the logic
   - Handles Python's negative modulo behavior
   - Efficient overflow detection using integer division bounds
   - Multiple implementation variants for learning

Highlights

- Overflow check: `result > INT_MAX // 10` prevents overflow before it happens
- Negative number handling: Correctly handles Python's modulo behavior for negative numbers
- Edge case coverage: Tests include boundary conditions and overflow scenarios

All tests pass, including edge cases near the 32-bit integer limits. The solution is ready for coding interviews and demonstrates efficient overflow handling.

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