AI ASSISTANT CODING

Assignment – 2.5

Name: N. Paul Benjamin

Roll Number: 2303A51116

Batch – 03

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

❖ Scenario:

You are improving legacy code.

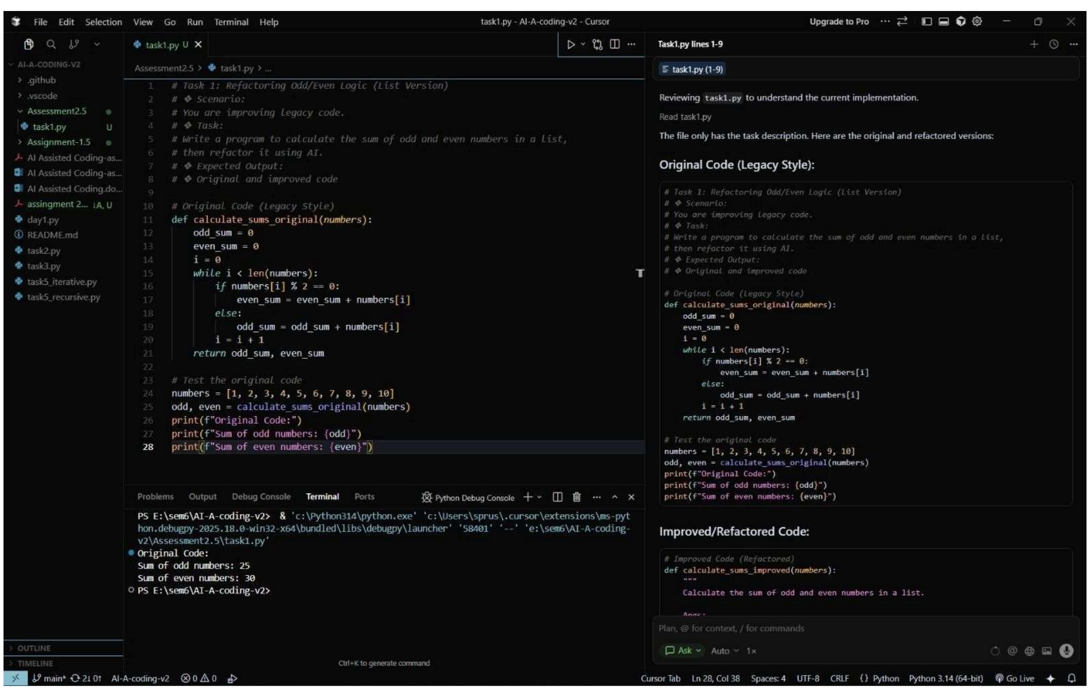
❖ Task:

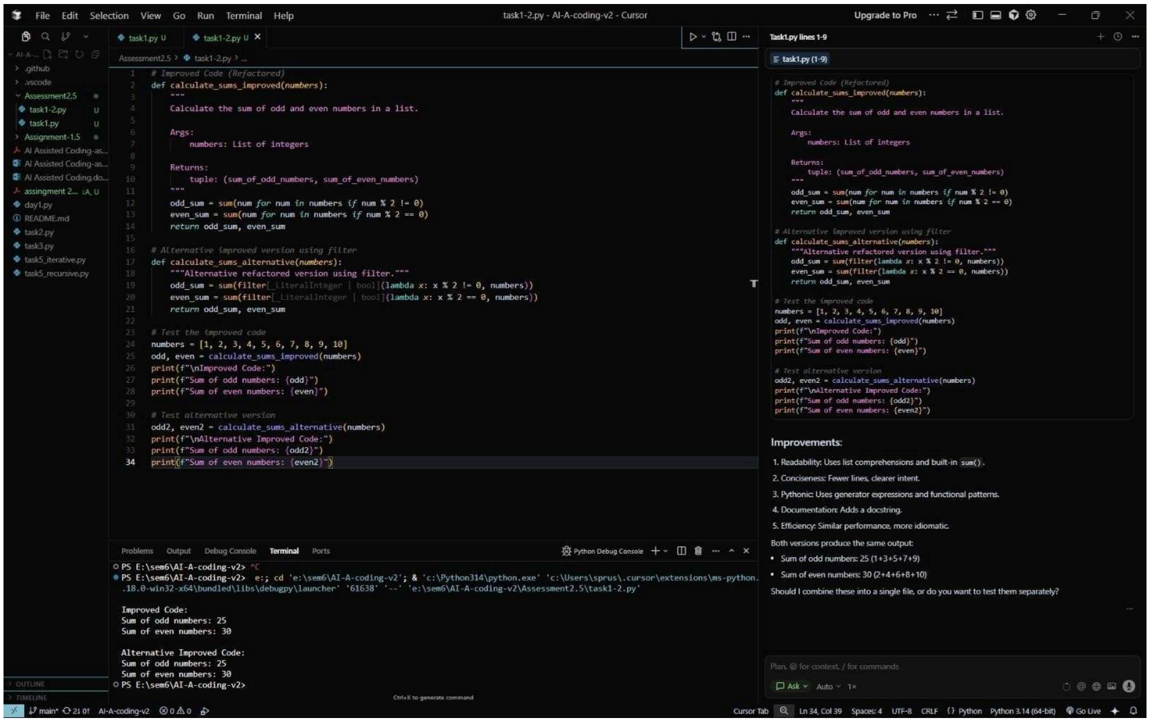
Write a program to calculate the sum of odd and even numbers in a list,

then refactor it using AI.

❖ Expected Output:

❖ Original and improved code





Task 2: Area Calculation Explanation

❖ Scenario:

You are onboarding a junior developer.

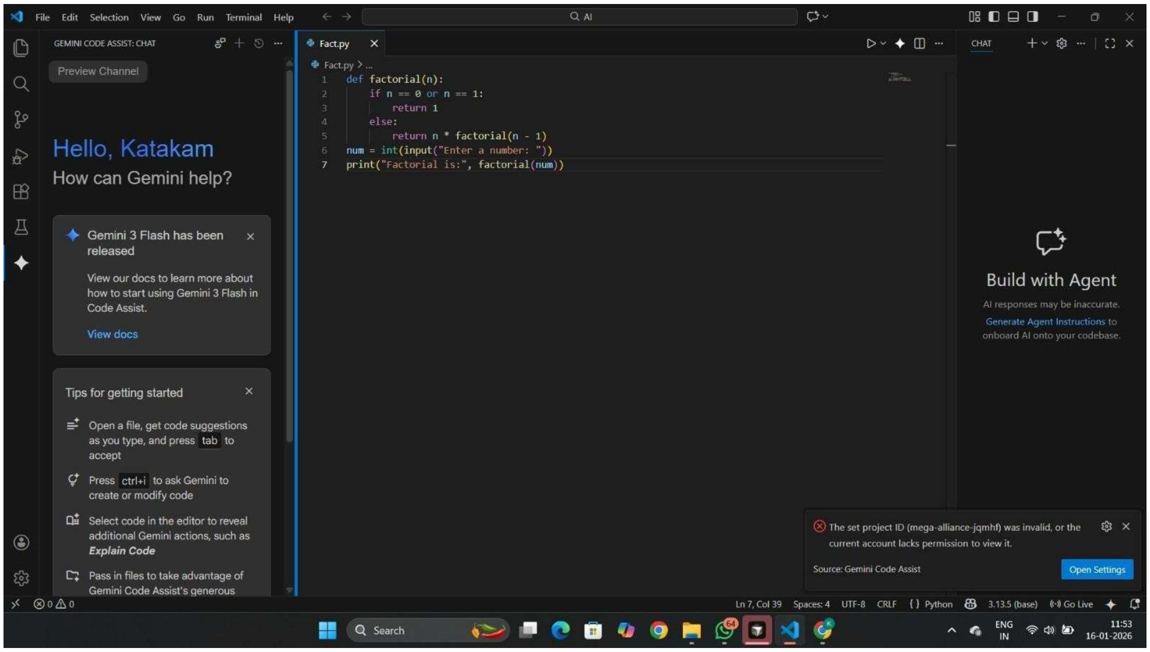
❖ Task:

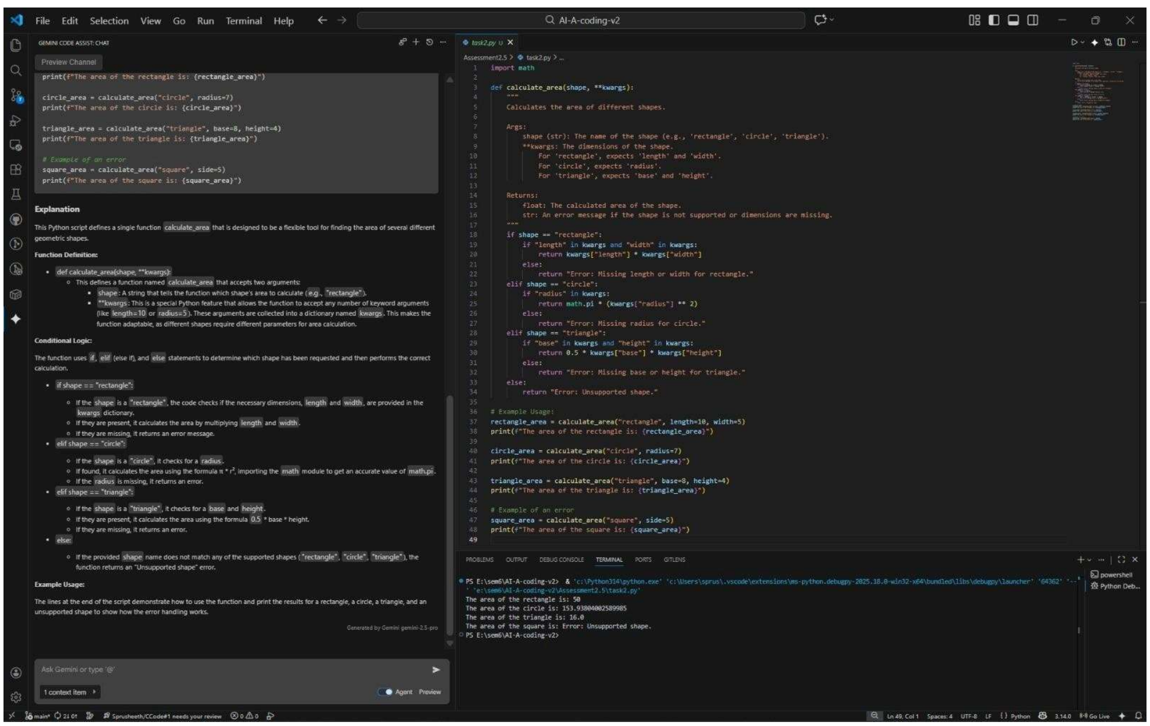
Ask Gemini to explain a function that calculates the area of different shapes.

❖ Expected Output:

➢ Code

➢ Explanation





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.

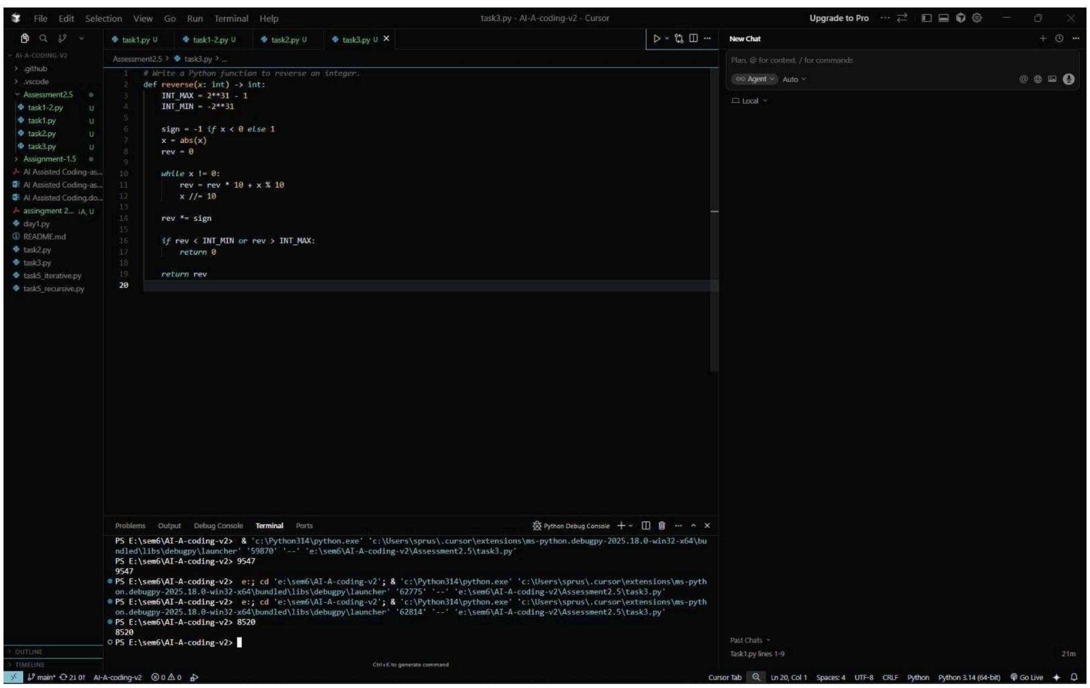
❖ **Task:**

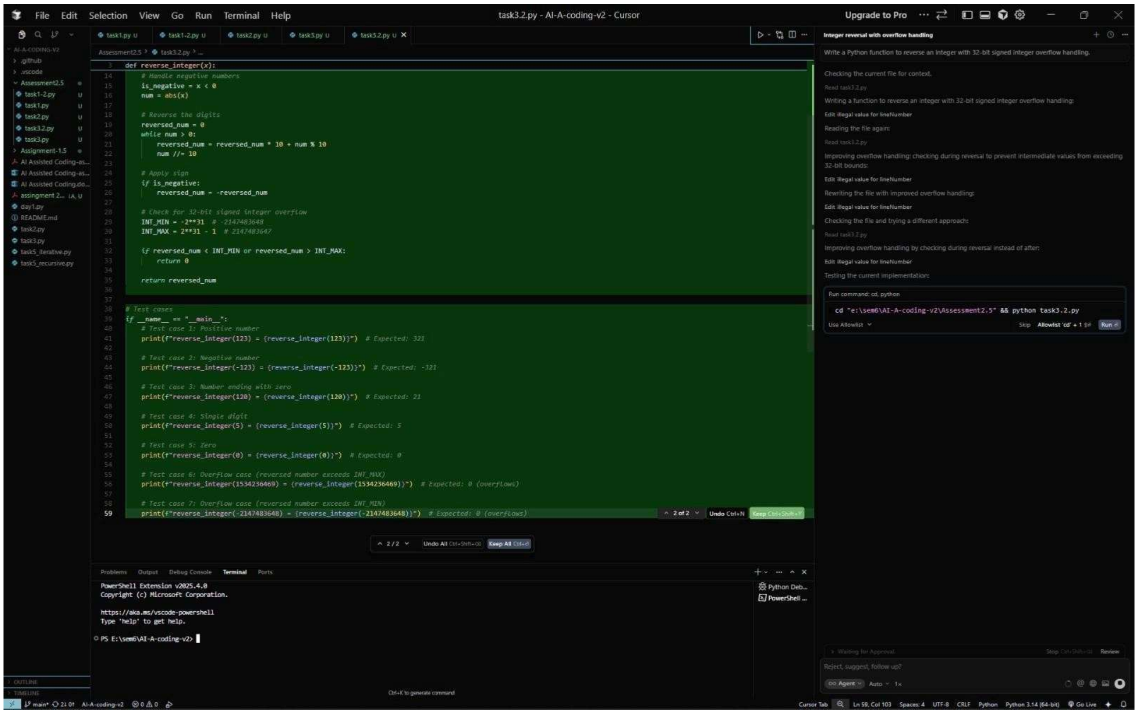
Use Cursor AI with different prompts for the same problem and observe code changes.

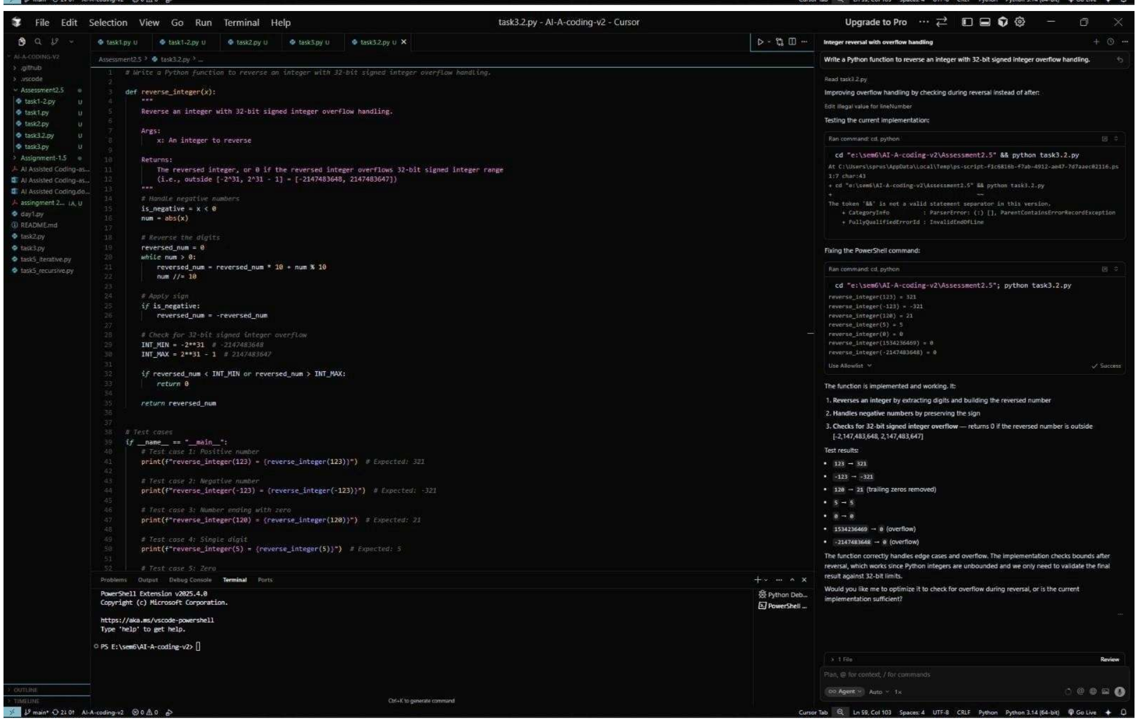
❖ **Expected Output:**

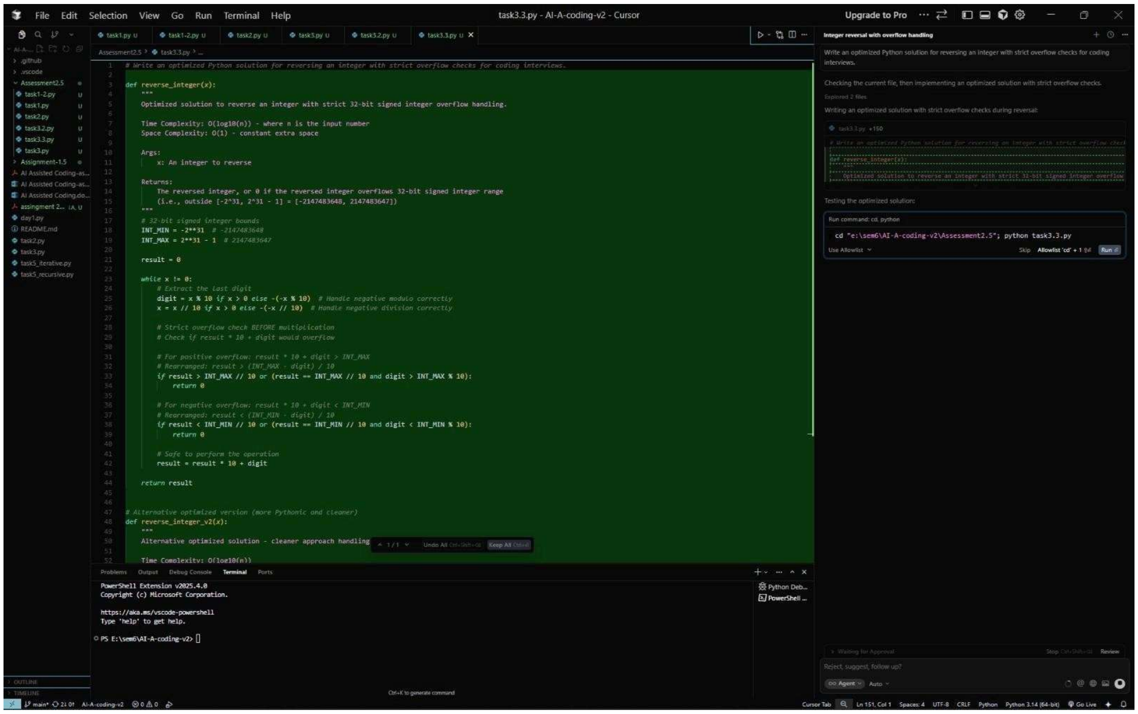
➢ **Prompt list**

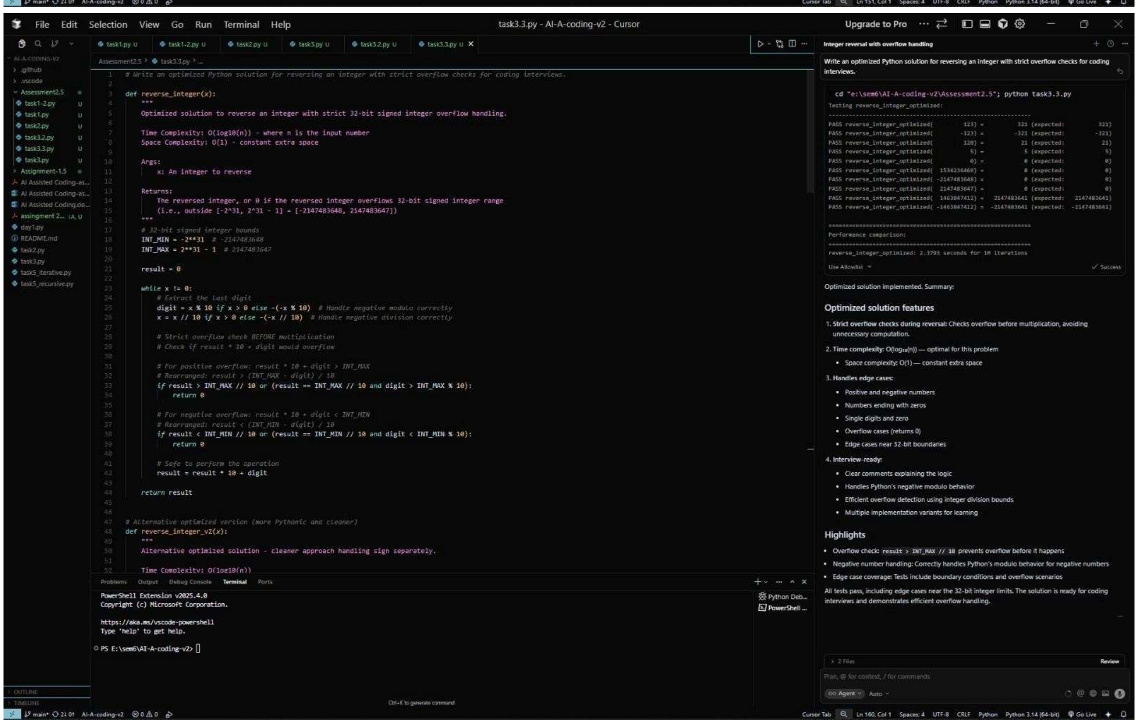
➢ **Code variations**




task3.2.py - AI-A-coding-v2 - Cursor
Integer reversal with overflow handling
Write a Python function to reverse an integer with 32-bit signed integer overflow handling.


task3.2.py - AI-A-coding-v2 - Cursor
Integer reversal with overflow handling
Write a Python function to reverse an integer with 32-bit signed integer 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