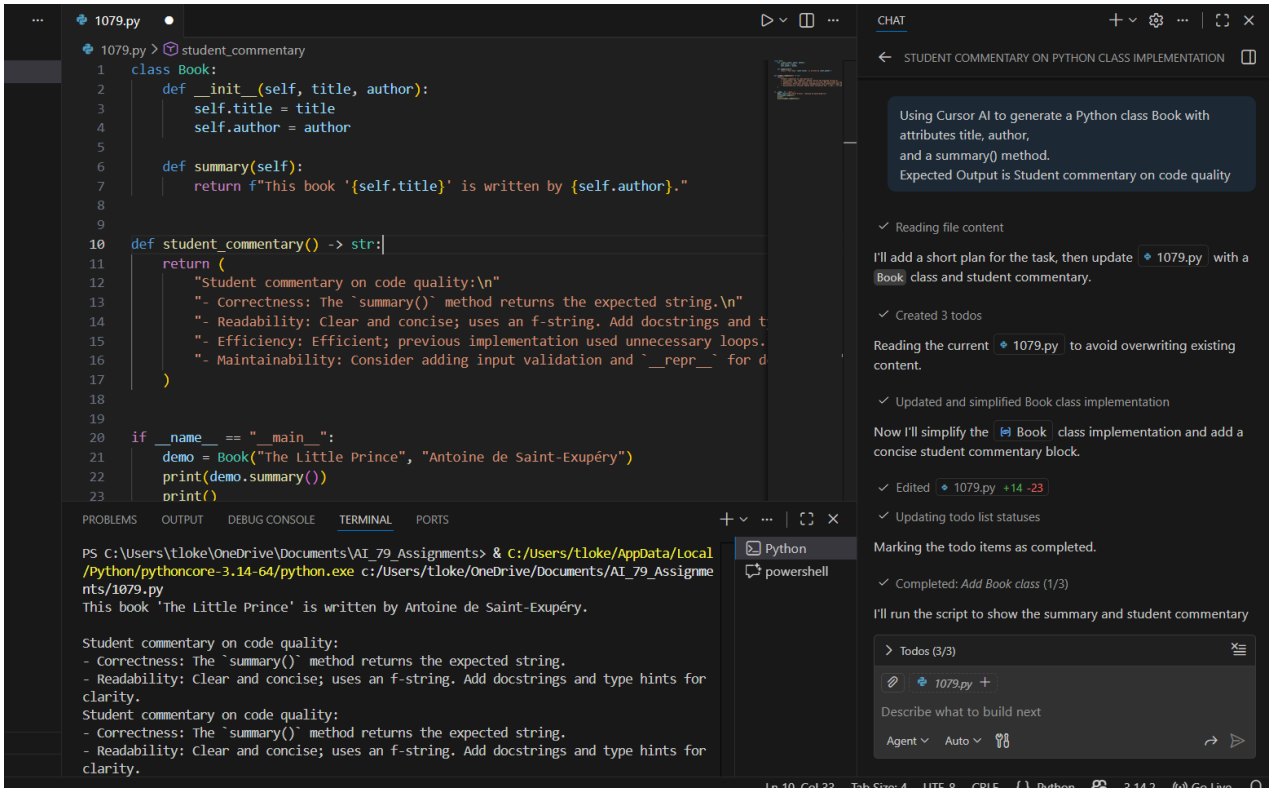
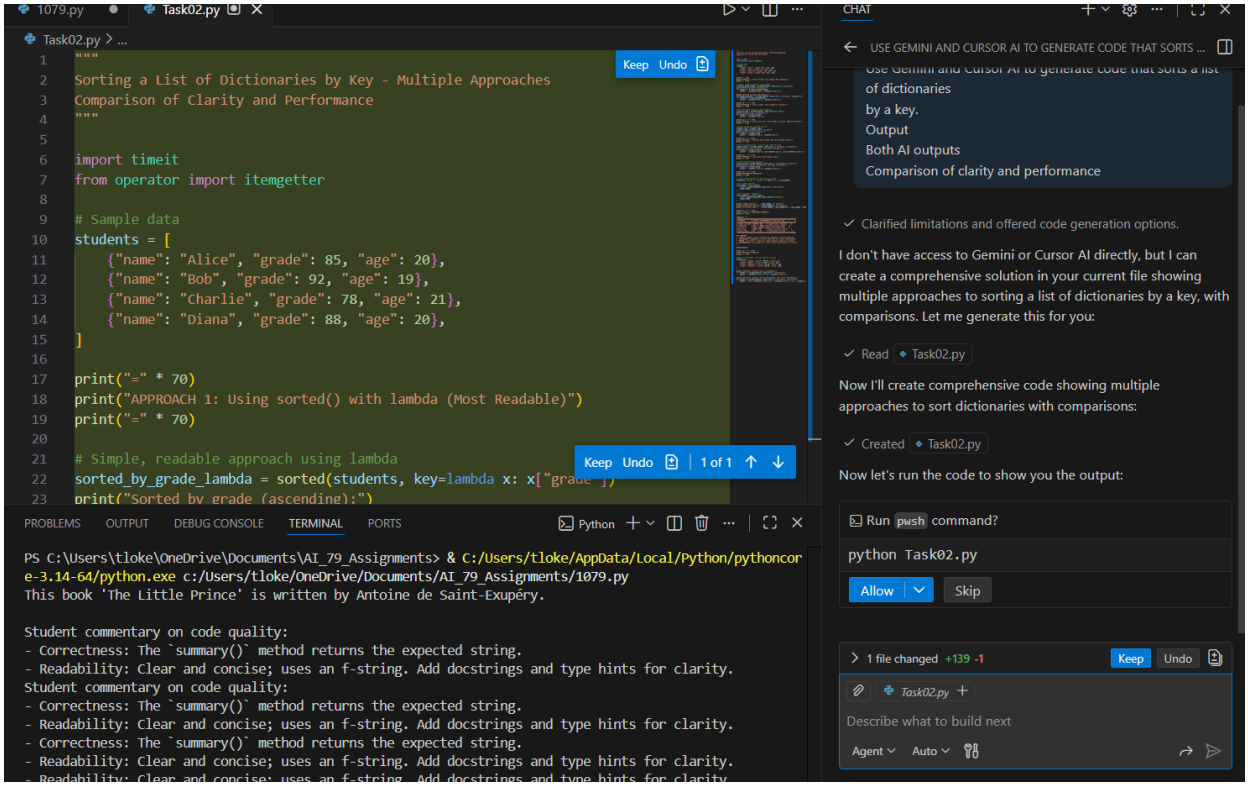
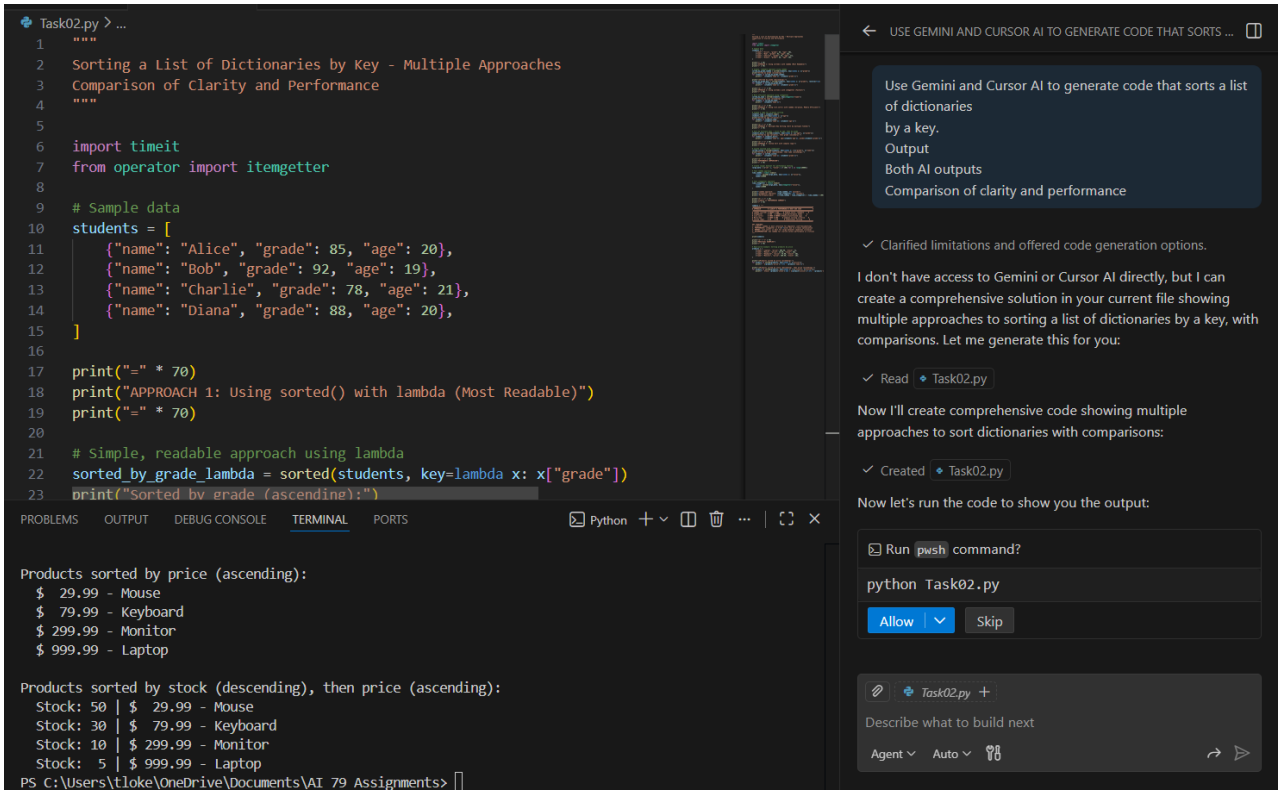
Task 1: Book Class Generation

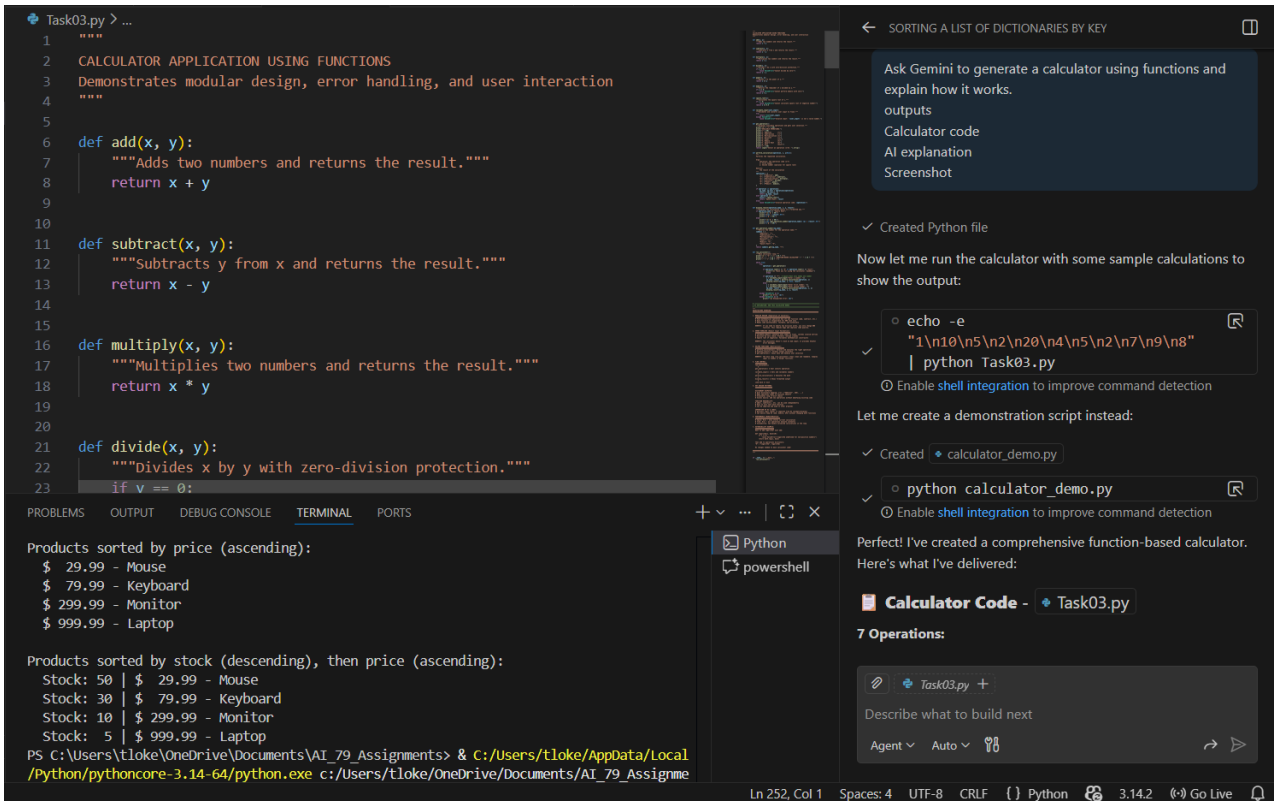


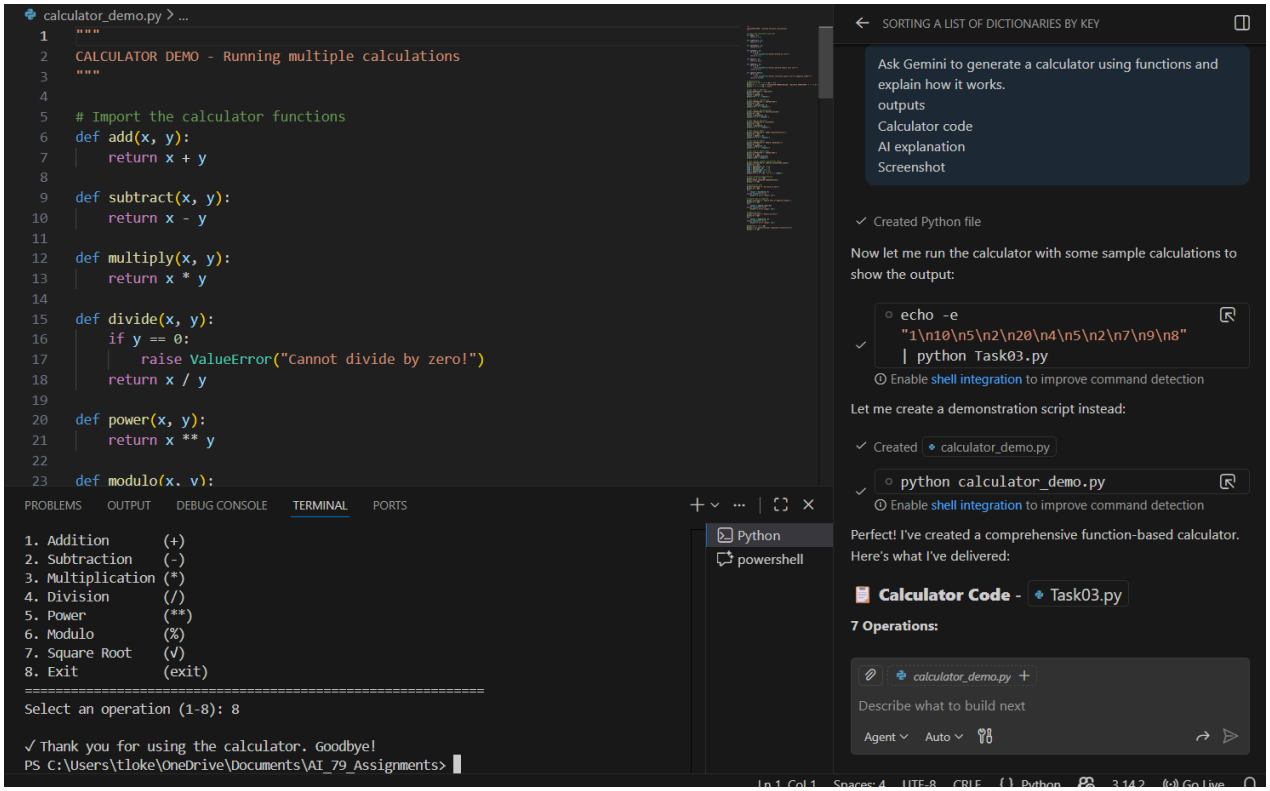
Task 2: Sorting Dictionaries with AI



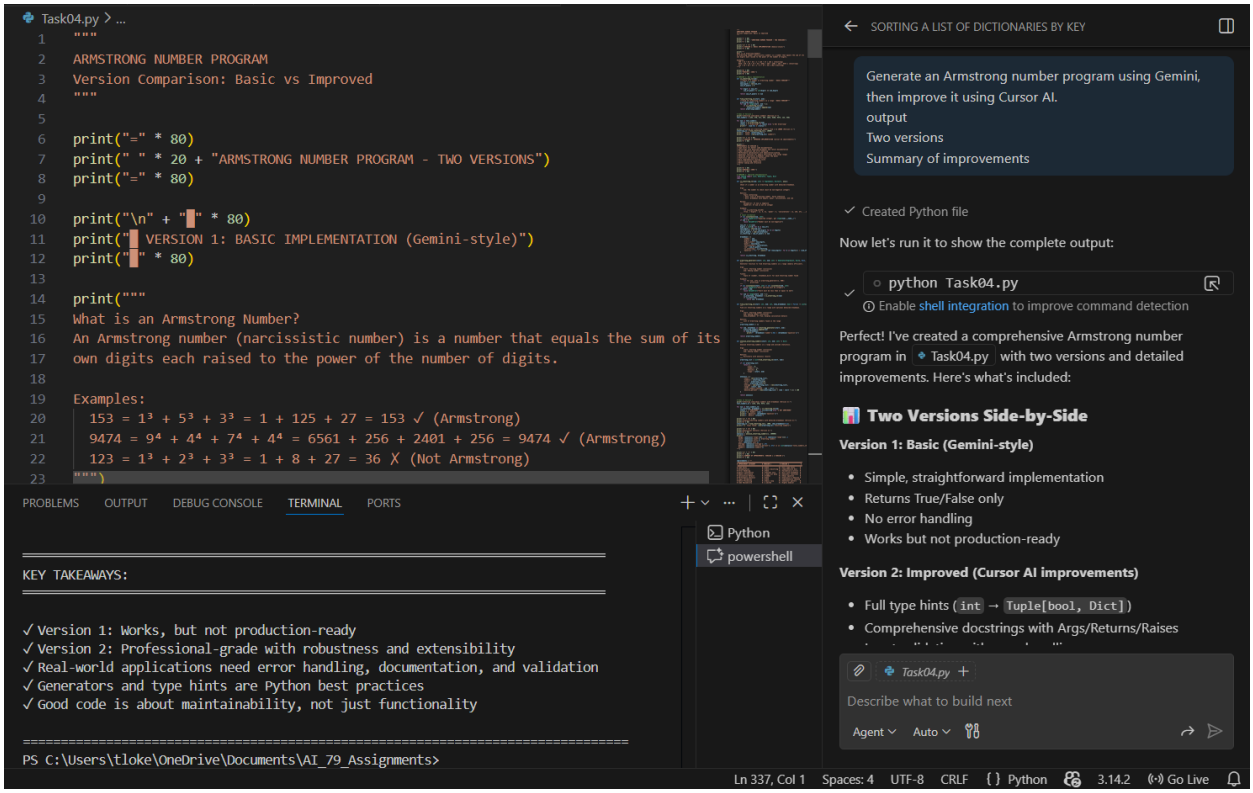


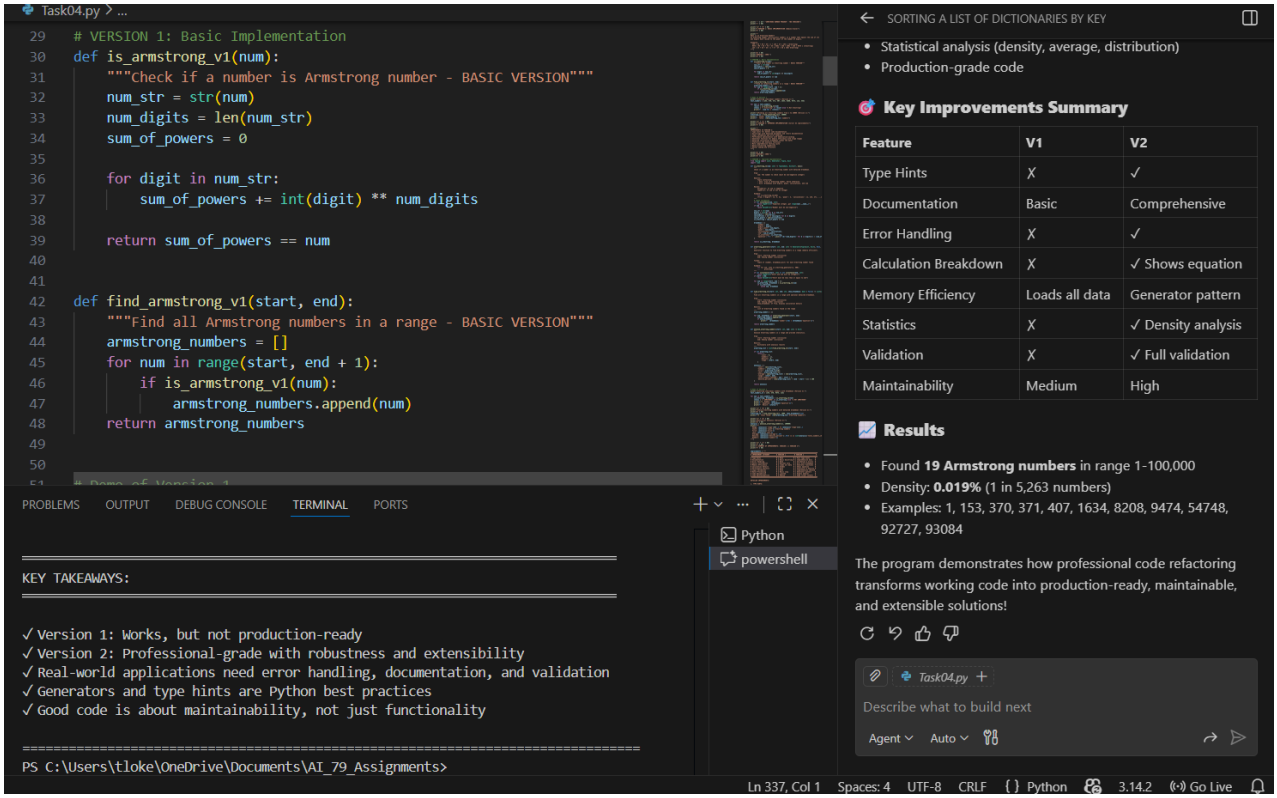
Task 3: Calculator Using Functions





Task 4: Armstrong Number Optimization




Task04.py > ...
# VERSION 1: Basic Implementation
def is_armstrong_v1(num):
"""Check if a number is Armstrong number - BASIC VERSION"""
num_str = str(num)
num_digits = len(num_str)
sum_of_powers = 0
for digit in num_str:
sum_of_powers += int(digit) ** num_digits
return sum_of_powers == num
def find_armstrong_v1(start, end):
"""Find all Armstrong numbers in a range - BASIC VERSION"""
armstrong_numbers = []
for num in range(start, end + 1):
if is_armstrong_v1(num):
armstrong_numbers.append(num)
return armstrong_numbers
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
powershell
KEY TAKEAWAYS:
✓ Version 1: Works, but not production-ready
✓ Version 2: Professional-grade with robustness and extensibility
✓ Real-world applications need error handling, documentation, and validation
✓ Generators and type hints are Python best practices
✓ Good code is about maintainability, not just functionality
PS C:\Users\tloke\OneDrive\Documents\AI_79_Assignments>
SORTING A LIST OF DICTIONARIES BY KEY
Statistical analysis (density, average, distribution)
Production-grade code
Key Improvements Summary
Feature
V1
V2
Type Hints
X
✓
Documentation
Basic
Comprehensive
Error Handling
X
✓
Calculation Breakdown
X
✓ Shows equation
Memory Efficiency
Loads all data
Generator pattern
Statistics
X
✓ Density analysis
Validation
X
✓ Full validation
Maintainability
Medium
High
Results
Found 19 Armstrong numbers in range 1-100,000
Density: 0.019% (1 in 5,263 numbers)
Examples: 1, 153, 370, 371, 407, 1634, 8208, 9474, 54748, 92727, 93084
The program demonstrates how professional code refactoring transforms working code into production-ready, maintainable, and extensible solutions!
Task04.py
Describe what to build next
Agent
Auto
Ln 337, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.14.2
Go Live