**AI ASSISTED CODING ASSIGNMENT 8.2**

**ENROLLMENT NO :**2503A51L21

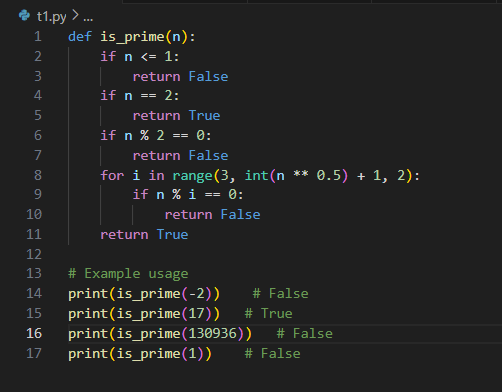
**BATCH NO:** 19

**NAME:** LUNGHIMPOU KAMEI

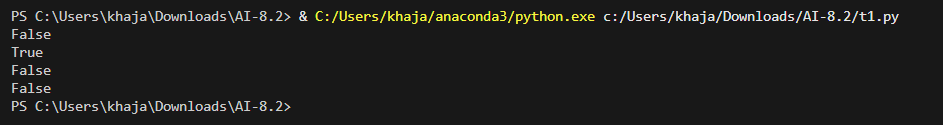
**TASK1:** Use AI to generate test cases for a function is\_prime(n) and then implement the function

**Requirements:**  
• Only integers > 1 can be prime.  
•Check edge cases: 0, 1, 2, negative numbers, and large primes

**PROMPT**: Generate a python function to test cases for a function is prime(n) and then implement the function with example.



**OUTPUT:**



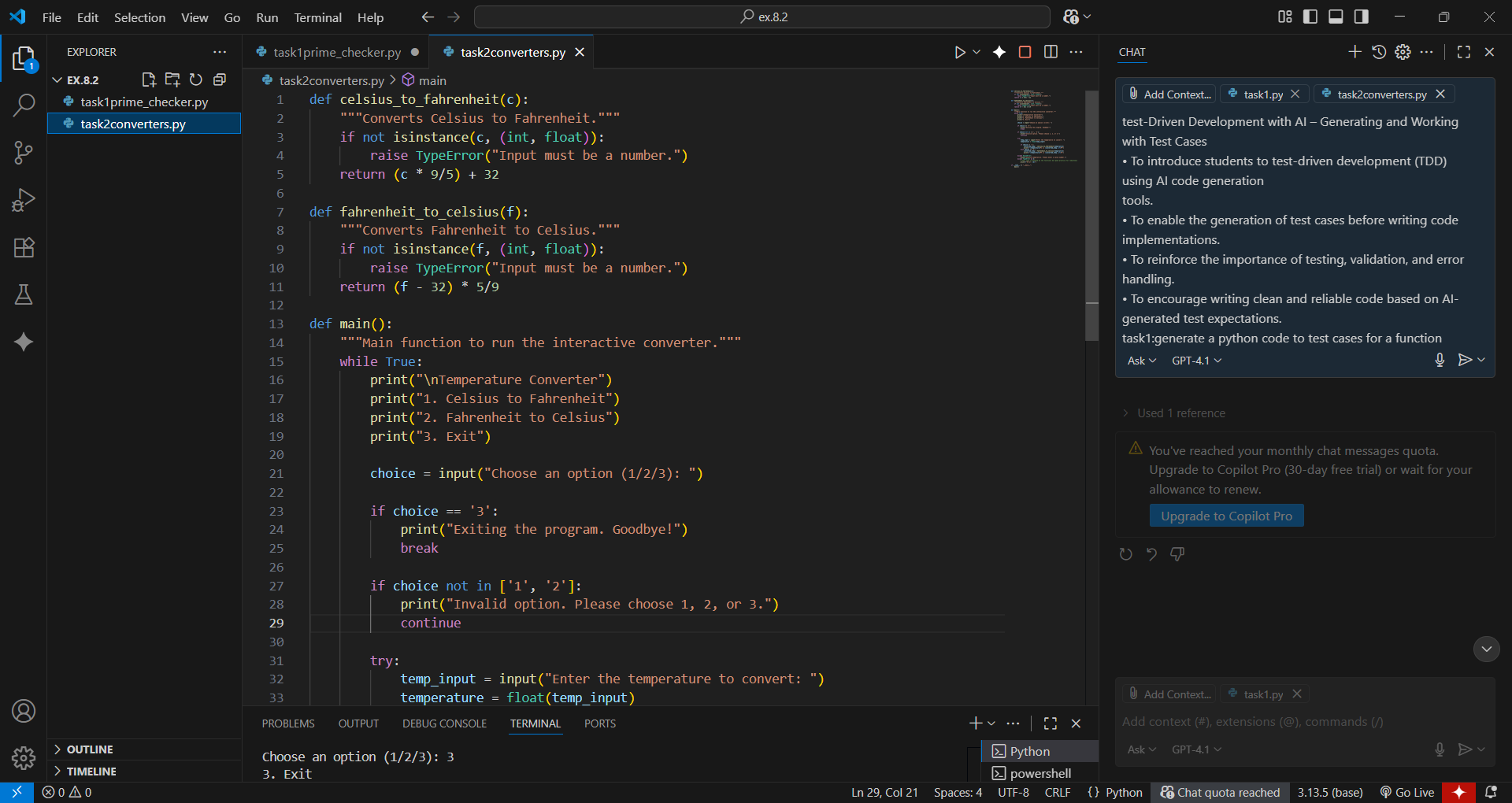
**OBSERVATION:**

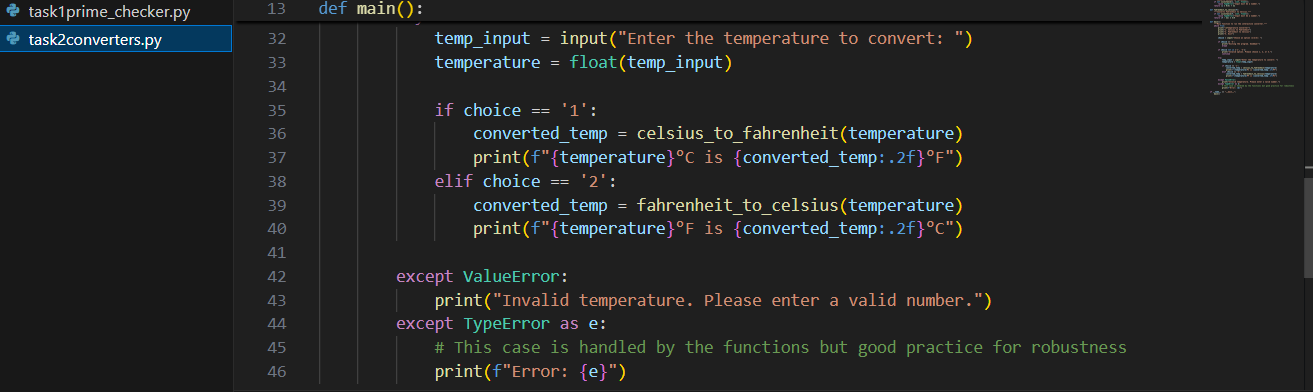
I observed that AI helped in generating test cases for edge inputs like 0, 1, negative numbers, and large primes. This ensured correctness of the is\_prime(n) function for both typical and boundary conditions. It showed how AI can reduce errors by considering cases a human might overlook.

**TASK2**: Ask AI to generate test cases for celsius\_to\_fahrenheit(c) and fahrenheit\_to\_celsius(f)

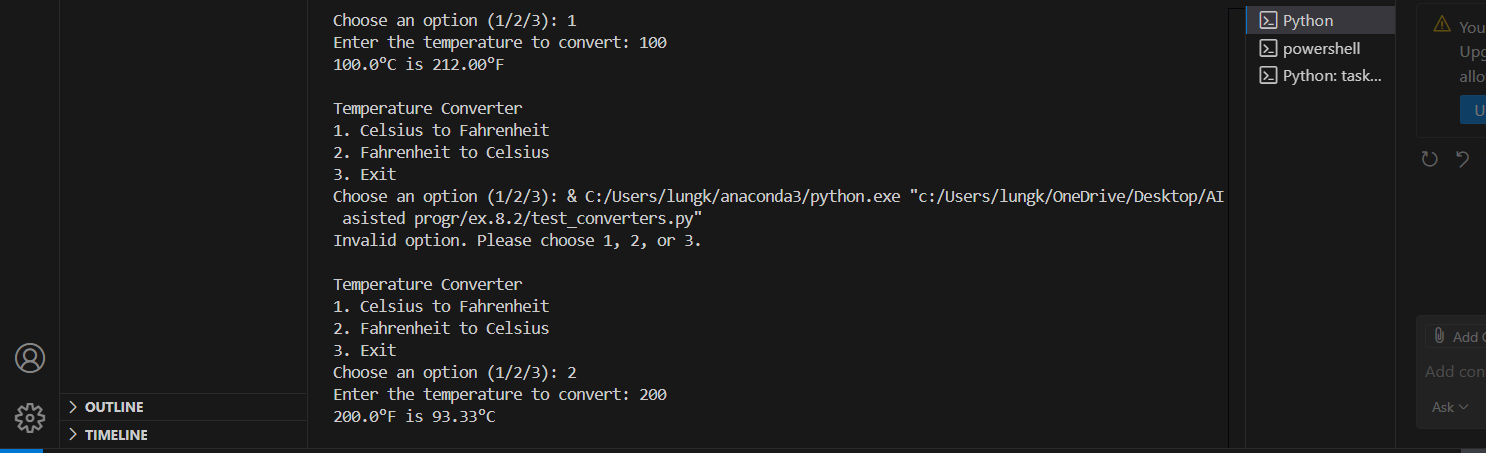
**Requirements**  
• Validate known pairs: 0°C = 32°F, 100°C = 212°F.  
• Include decimals and invalid inputs like strings or None

**PROMPT**: Generate a python code to test cases for celsius\_to\_fahrenheit(c) and fahrenheit\_to\_celsius(f).

****



OUTPUT:



**OBSERVATION:**

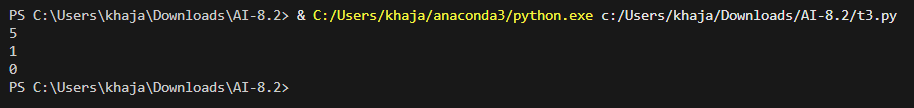
I observed that AI-generated test cases validated both standard conversions (e.g., 0°C = 32°F) and unusual inputs like decimals and invalid types (None, strings). This improved the reliability of the conversion functions and highlighted the importance of including negative and floating-point values.

**TASK3**: Use AI to write test cases for a function count\_words(text) that returns the number of words in a sentence.  
**Requirement**  
Handle normal text, multiple spaces, punctuation, and empty strings

**PROMPT**: Write test cases for a function count\_words(text) that returns the number of  
words in a sentence

****

**OUTPUT:**

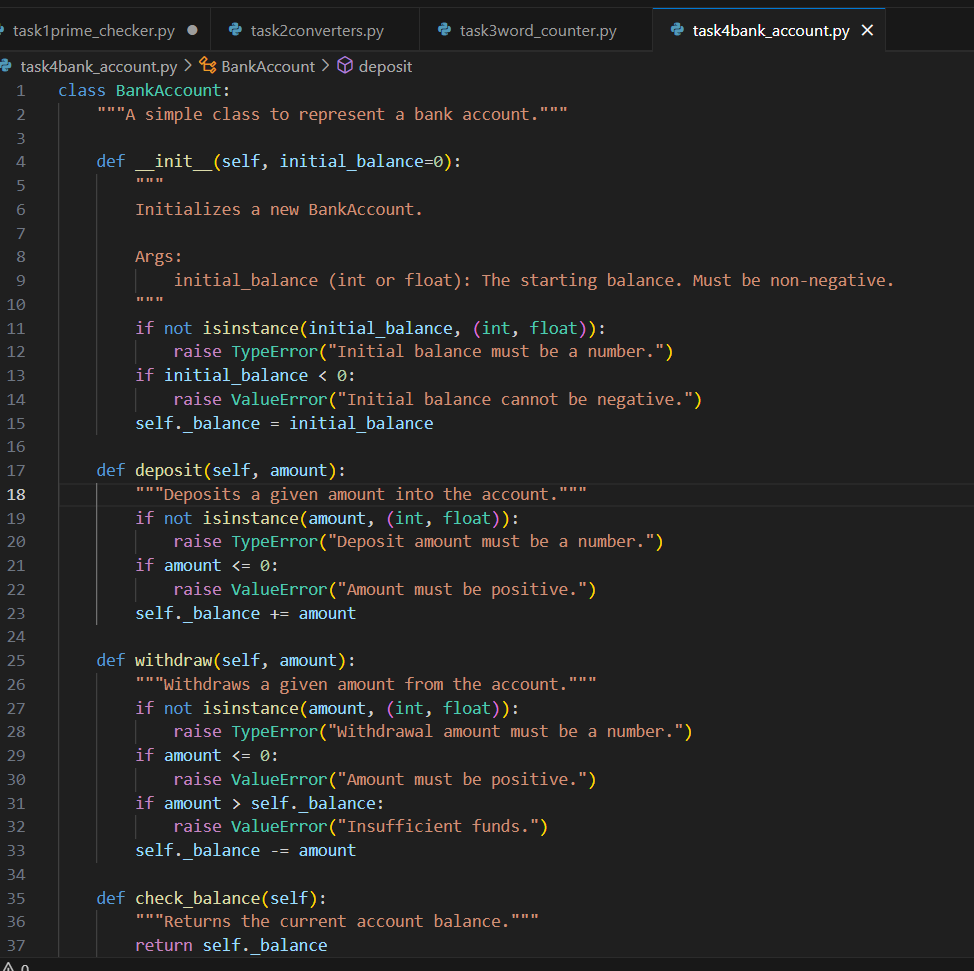
****

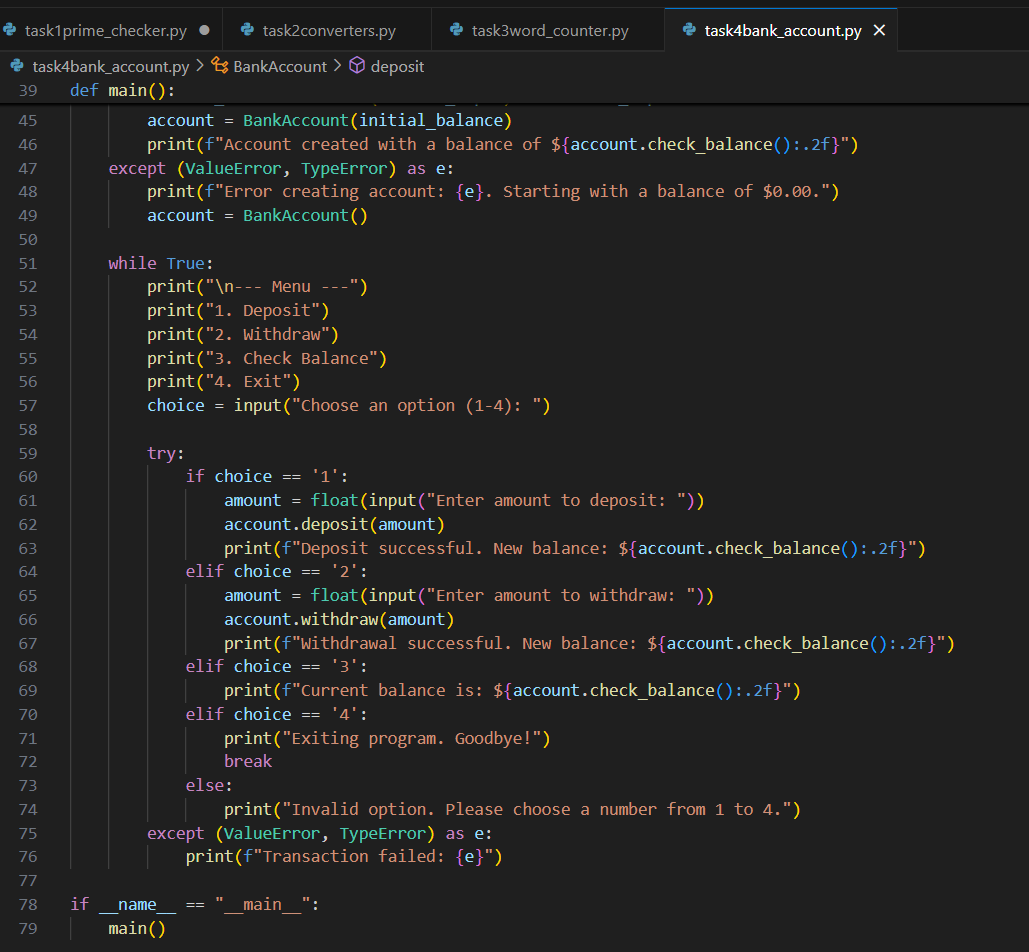
**OBSERVATION:**

I observed that the test cases covered normal sentences, multiple spaces, punctuation, and empty strings. This demonstrated how AI can anticipate real-world scenarios where input formatting varies. It showed the importance of testing beyond ideal cases.

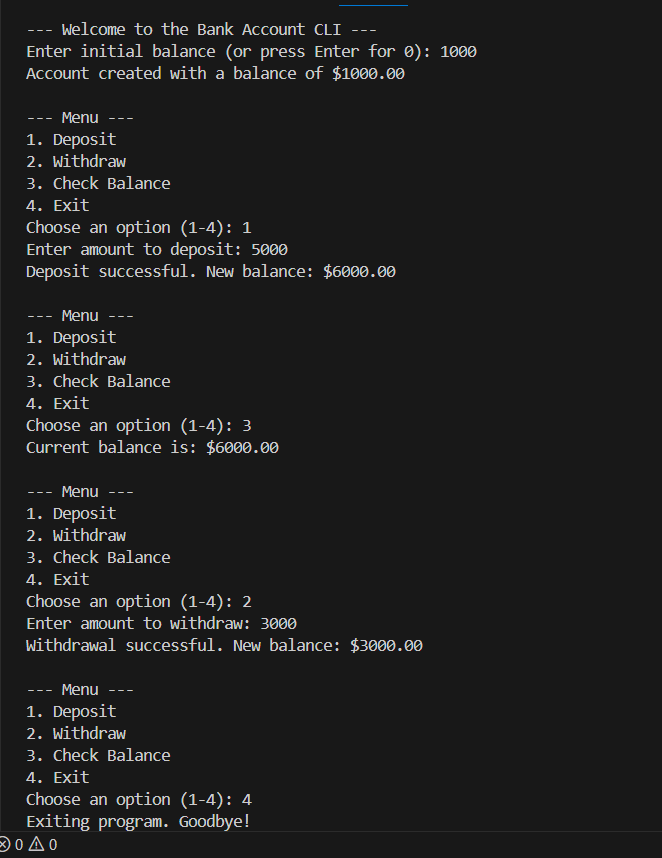
**TASK4:** Generate test cases for a BankAccount class with:  
**Methods:**  
deposit(amount)  
withdraw(amount)  
check\_balance ()  
**Requirements:**  
• Negative deposits/withdrawals should raise an error.  
• Cannot withdraw more than balance

**PROMPT:** Generate test cases for a BankAccount class with Methods:deposit(amount),withdraw(amount),check\_balance()

****

****

OUTPUT:



OBSERVATION:

1. The BankAccount class encapsulates account operations with proper **data validation** for balance, deposits, and withdrawals.

2. The program handles **invalid input gracefully**, raising TypeError or ValueError when the user enters non-numeric values, negative numbers, or tries to withdraw more than the balance.

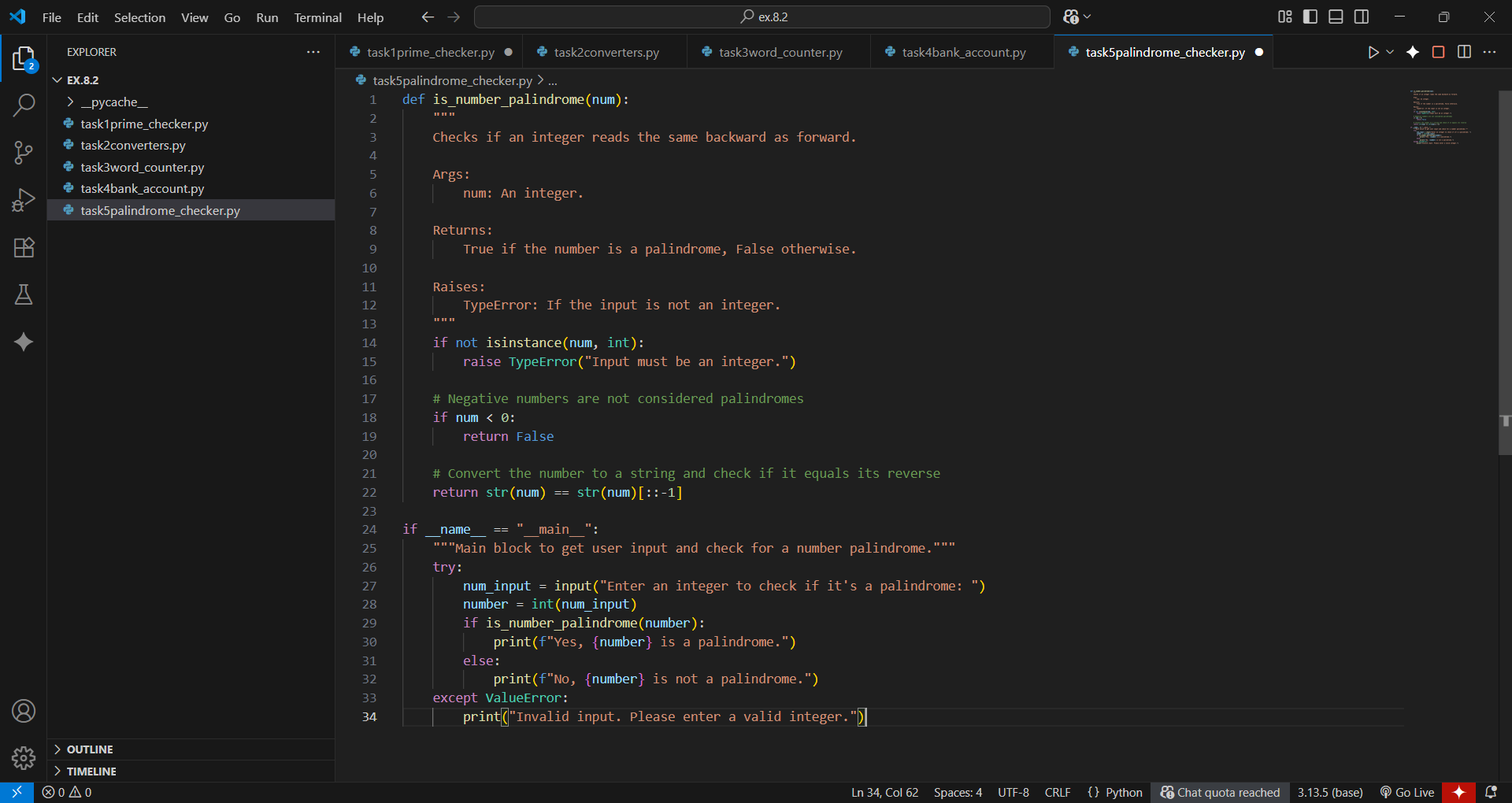
3. The CLI menu allows users to **deposit**, **withdraw**, **check balance**, or **exit**, looping until exit.

4. The program ensures **robustness** by catching exceptions during account creation and transactions, providing clear messages without crashing.

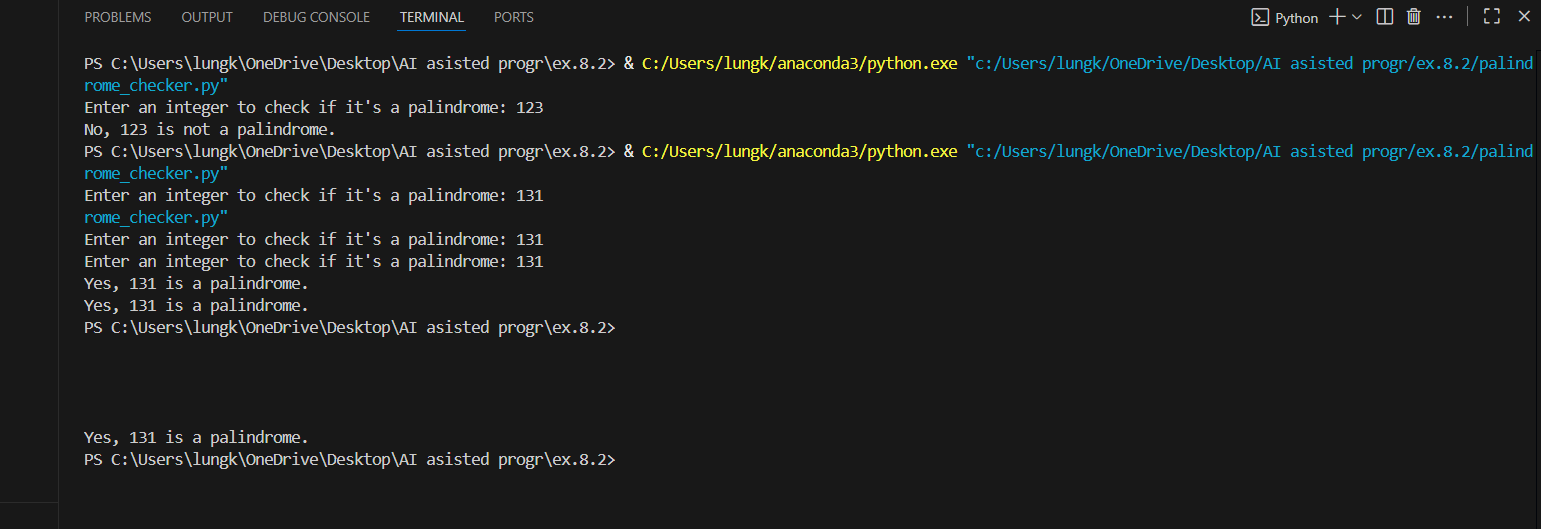
5. The balance is displayed with **two decimal places**, making it realistic for a financial application.

**TASK5:** Generate test cases for is\_number\_palindrome(num), which checks if an integer reads the same backward.  
**Examples:**  
121 → True  
123 → False  
0, negative numbers → handled gracefully

**PROMPT:** Write a Python function to test cases for is\_number\_palindrome(num), which checks if an integer reads the same backward

****

**OUTPUT:**



**OBSERVATION:** The function is\_number\_palindrome(num) first validates that the input is an integer, raising a TypeError if not. Negative integers are treated as non-palindromes. The function converts the number to a string and compares it to its reverse to determine if it reads the same backward and forward. The main block handles user input, safely converts it to an integer, and prints a message indicating whether the number is a palindrome. It also gracefully handles invalid inputs using a try-except block, preventing runtime errors.