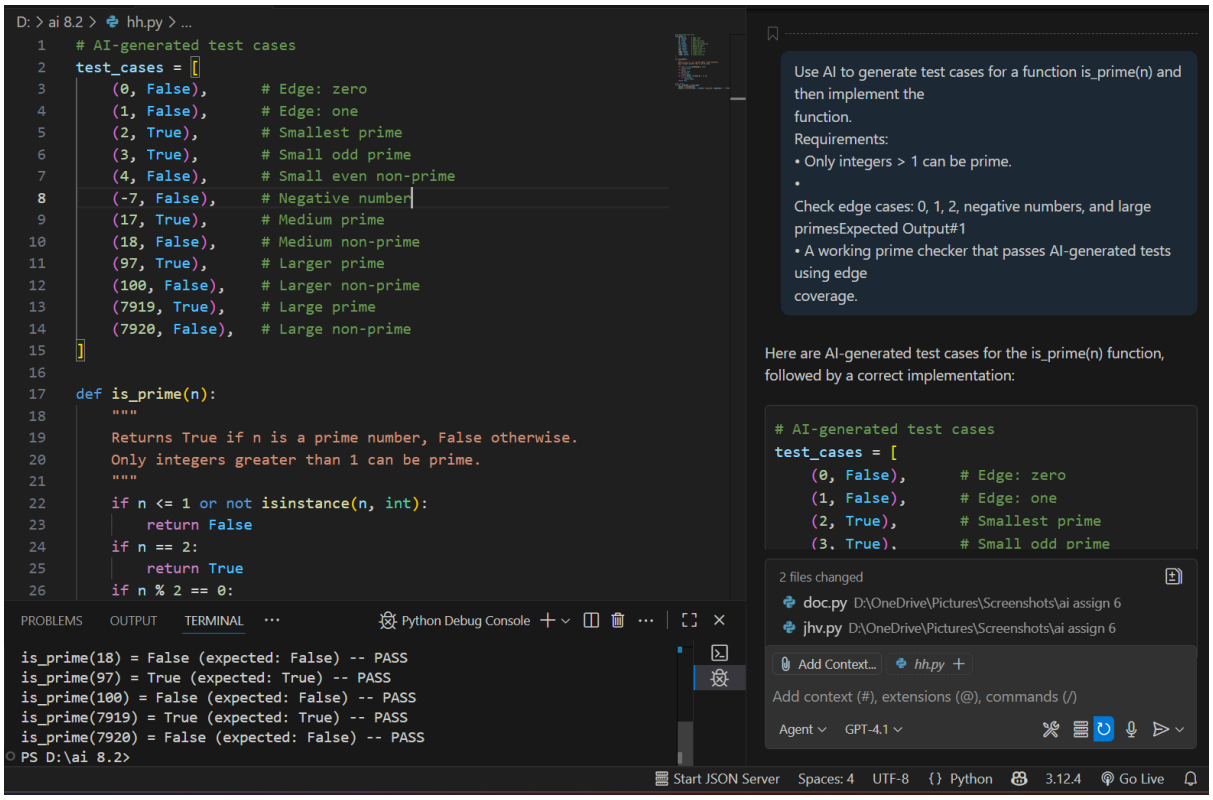
NAME :B.RAJAGOPAL

Roll No :NAME :2503A51L01

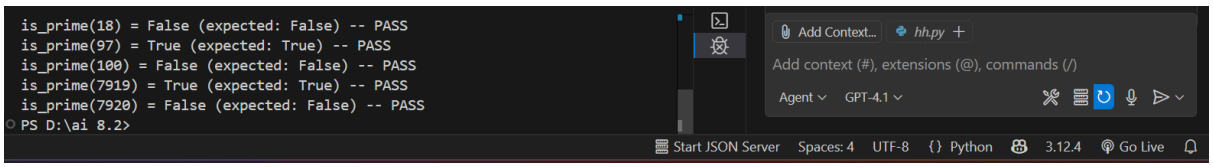
Batch : CSB19

TASK 1

Use AI to generate test cases for a function is_prime(n) and then implement the function



OUTPUT



OBSERVATION

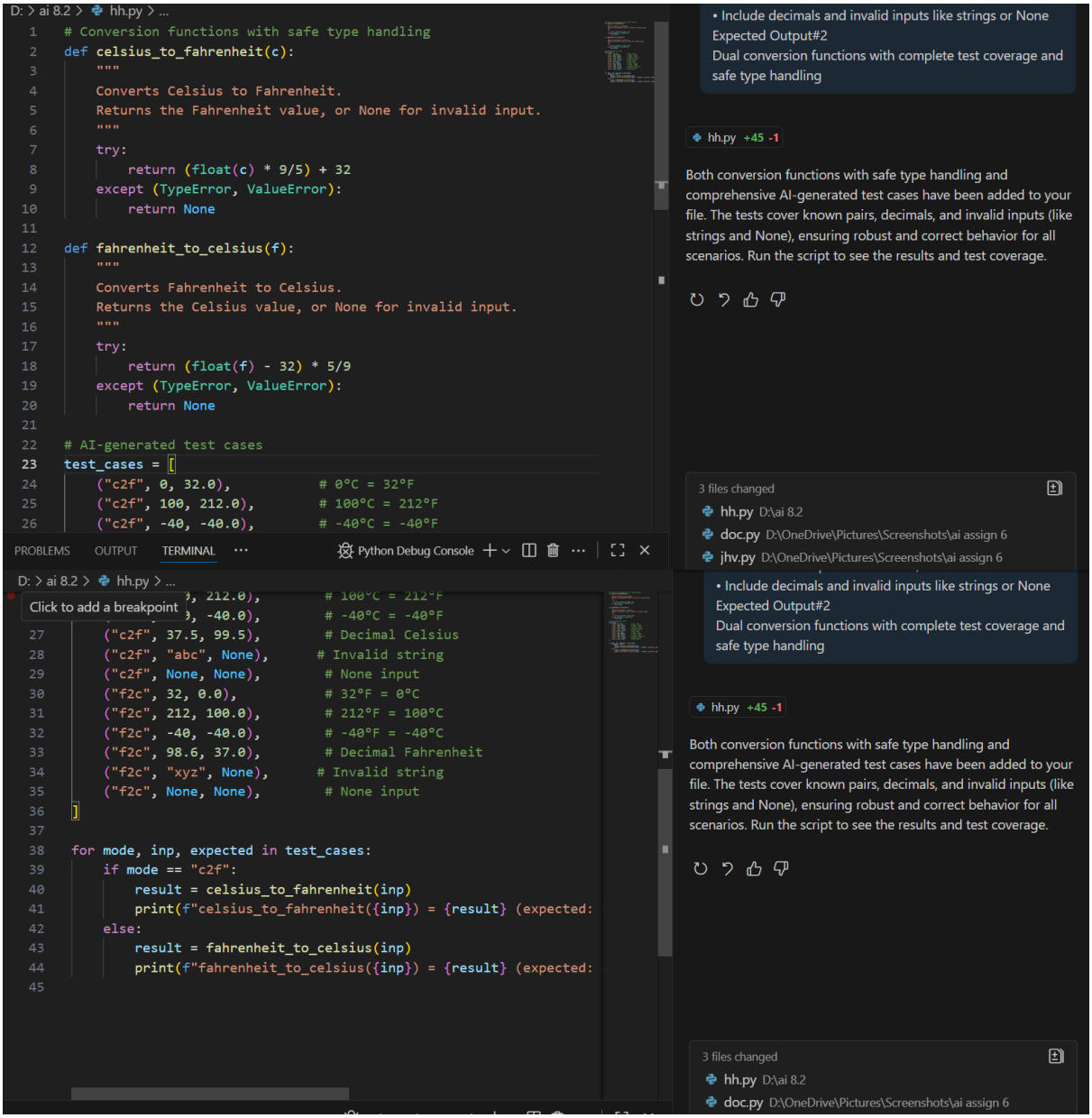
- The is_prime(n) function checks if a number is prime.
    - It returns False for any number less than or equal to 1, or if the input is not an integer.
    - It returns True for 2 (the smallest prime).

    - It returns False for even numbers greater than 2.
    - For odd numbers greater than 2, it checks divisibility from 3 up to the square root of n. If any divisor is found, it returns False; otherwise, it returns True.
- The test_cases list contains pairs of input values and their expected results, covering edge cases like 0, 1, negative numbers, small and large primes, and non-primes.
- The test loop runs is_prime on each test case and prints whether the result matches the expected value, helping verify the correctness of the function.

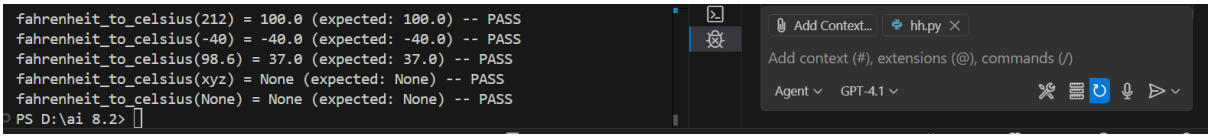
TASK2

Ask AI to generate test cases for celsius_to_fahrenheit(c) and fahrenheit_to_celsius(f).

CODE



OUTPUT



OBSERVATION

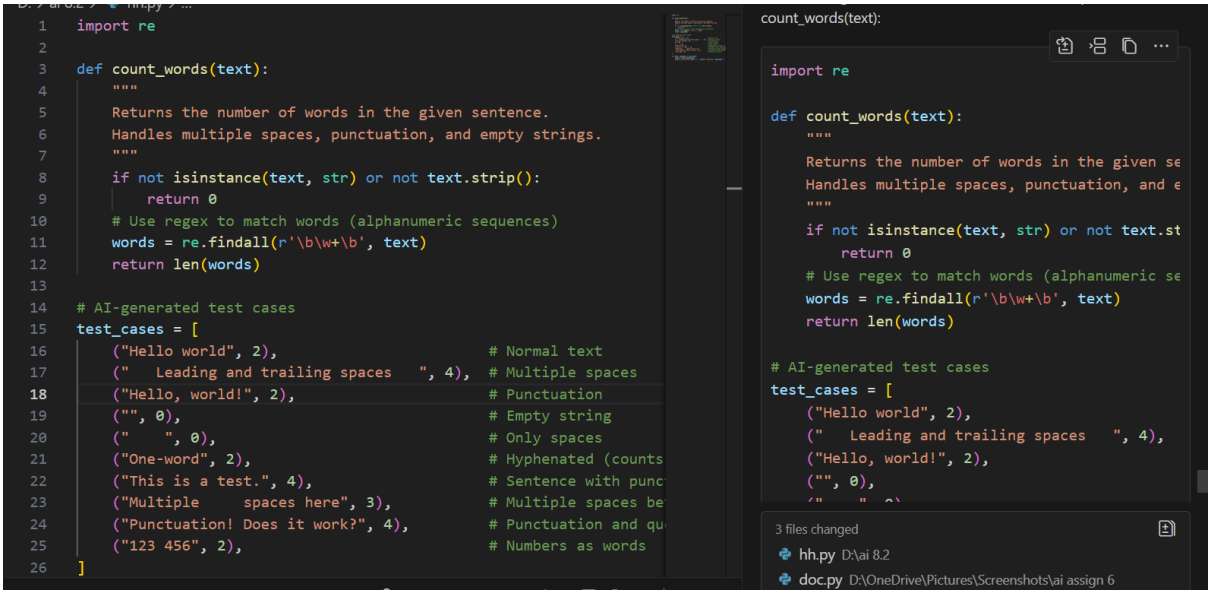
- Two conversion functions are defined:
    - celsius_to_fahrenheit(c): Converts a Celsius value to Fahrenheit. It safely handles invalid input (like strings or None) by returning None if conversion fails.
    - fahrenheit_to_celsius(f): Converts a Fahrenheit value to Celsius, also returning None for invalid input.
- A list of AI-generated test cases (test_cases) covers:
    - Known conversion pairs (e.g., 0°C = 32°F, 100°C = 212°F, -40°C = -40°F).

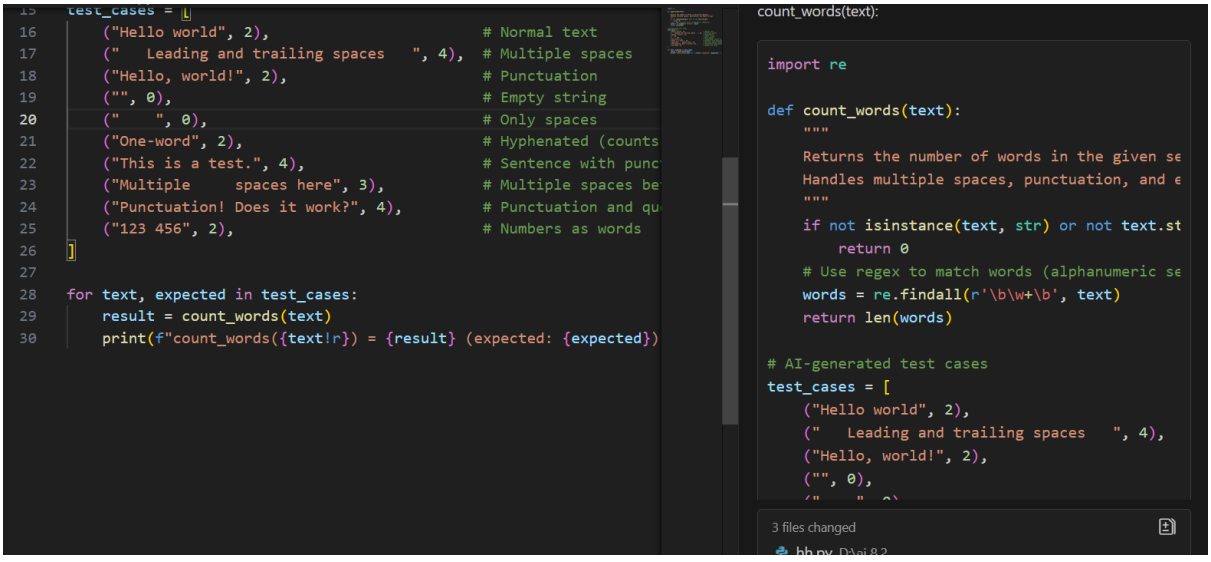
    - Decimal values (e.g., 37.5°C, 98.6°F).
    - Invalid inputs (e.g., strings like "abc" or "xyz", and None).
- The code iterates through each test case, calls the appropriate function, and prints the result along with whether it matches the expected output ("PASS" or "FAIL").
- This approach ensures both functions are robust, handle edge cases, and are validated against a comprehensive set of test scenarios**.**

TASK 3

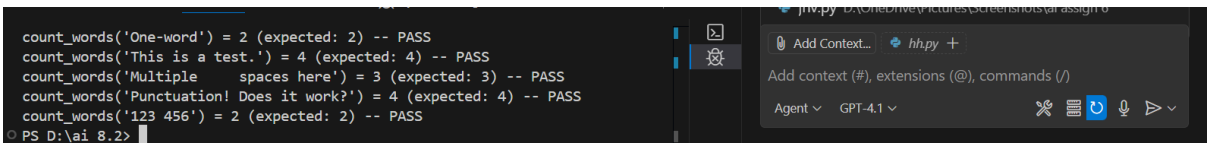
Use AI to write test cases for a function count_words(text) that returns the number of words in a sentence

CODE





OUTPUT



OBSERVATION

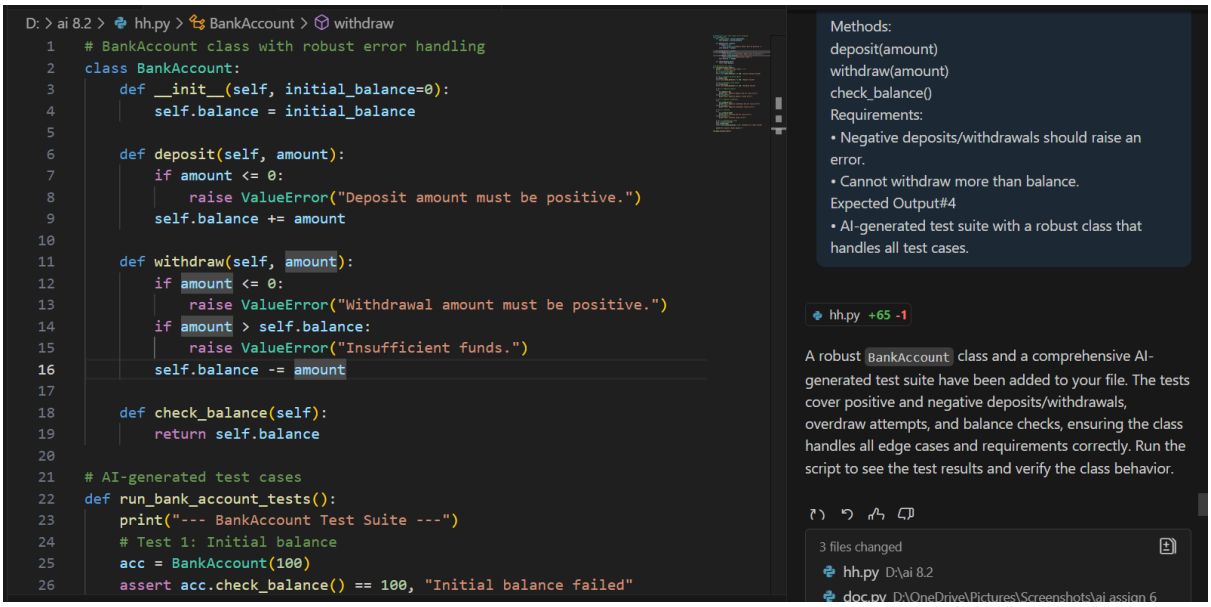
The **count_words** function accurately counts the number of words in a sentence, handling normal text, multiple spaces, punctuation, and empty or whitespace-only strings. The AI-

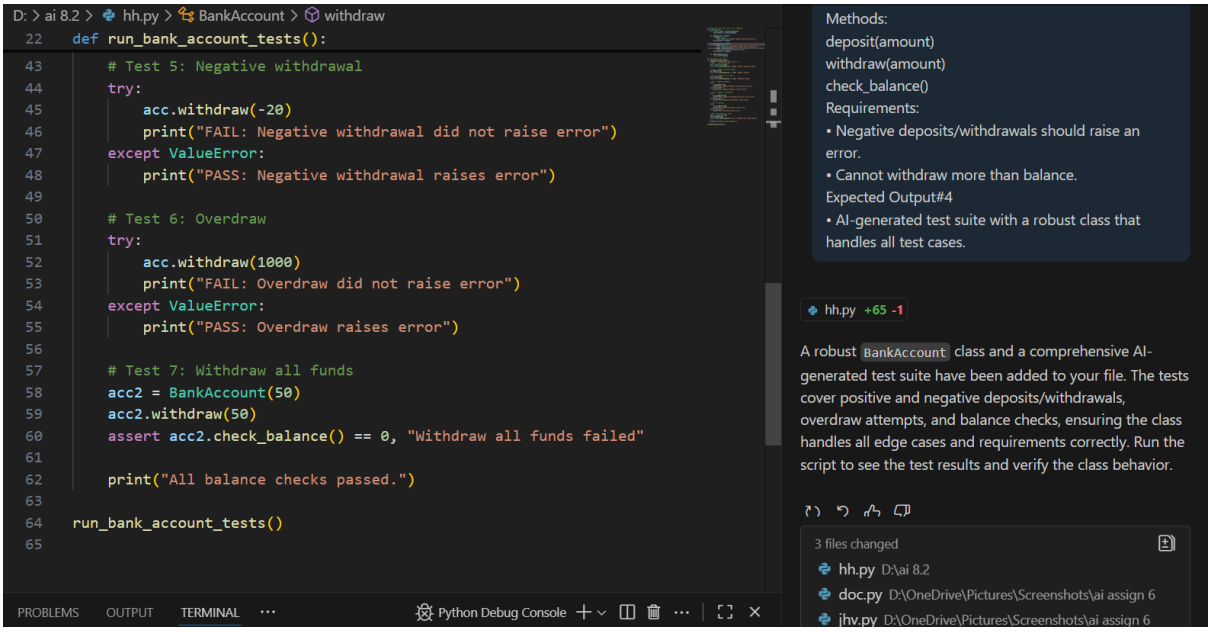
generated test cases comprehensively validate the function across various scenarios, including edge cases. This ensures the implementation is robust and reliable for different types of input, demonstrating the effectiveness of using AI to generate thorough test coverage for text-processing functions.

TASK 4

Generate test cases for a BankAccount class with:
Methods:
deposit(amount)
withdraw(amount)
check_balance()

CODE





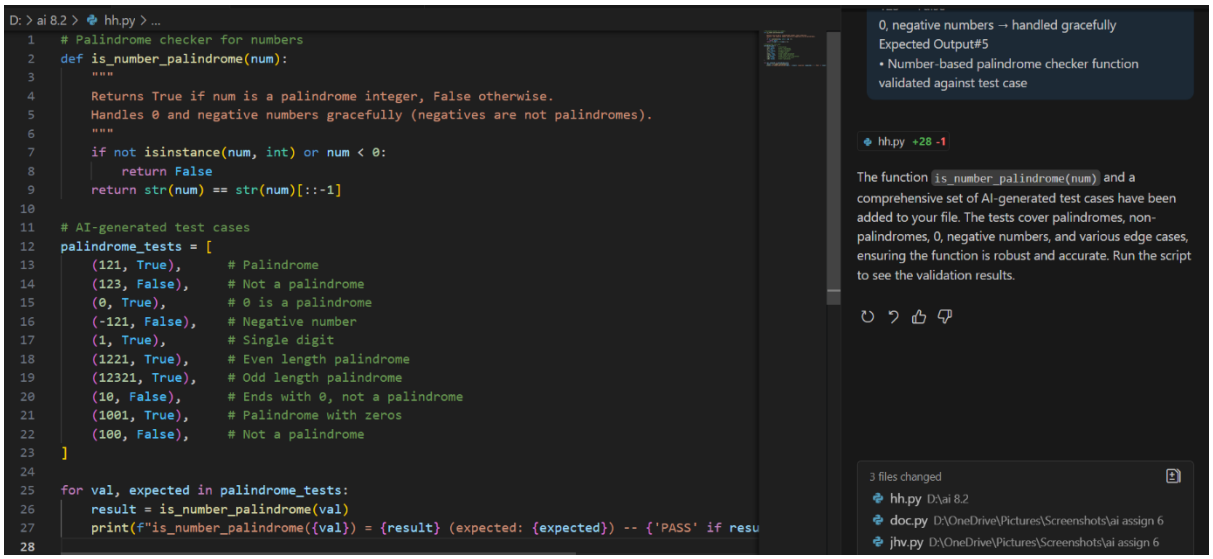
OUTPUT

OBSERVATION

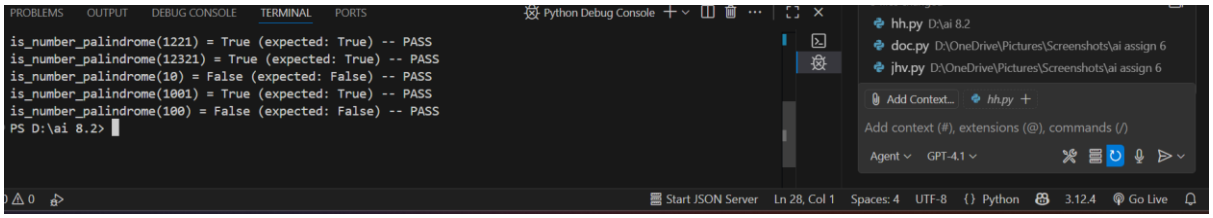
The BankAccount class is robustly designed to handle deposits, withdrawals, and balance checks, with safeguards against negative transactions and overdrafts. The AI-generated test suite thoroughly validates the class by checking normal operations, edge cases, and error conditions. This ensures the class behaves correctly and securely in all scenarios, demonstrating the effectiveness of comprehensive automated testing for financial logic.

TASK 5

Generate test cases for is_number_palindrome(num), which checks if an integer reads the same backward



OUTPUT



OBSERVATION

The is_number_palindrome function correctly determines whether an integer reads the same backward, handling edge cases such as 0, negative numbers, and single digits. The AI-generated test suite thoroughly validates the function across a variety of scenarios, ensuring reliable and accurate results. This demonstrates the value of comprehensive test coverage and robust input handling in utility functions.