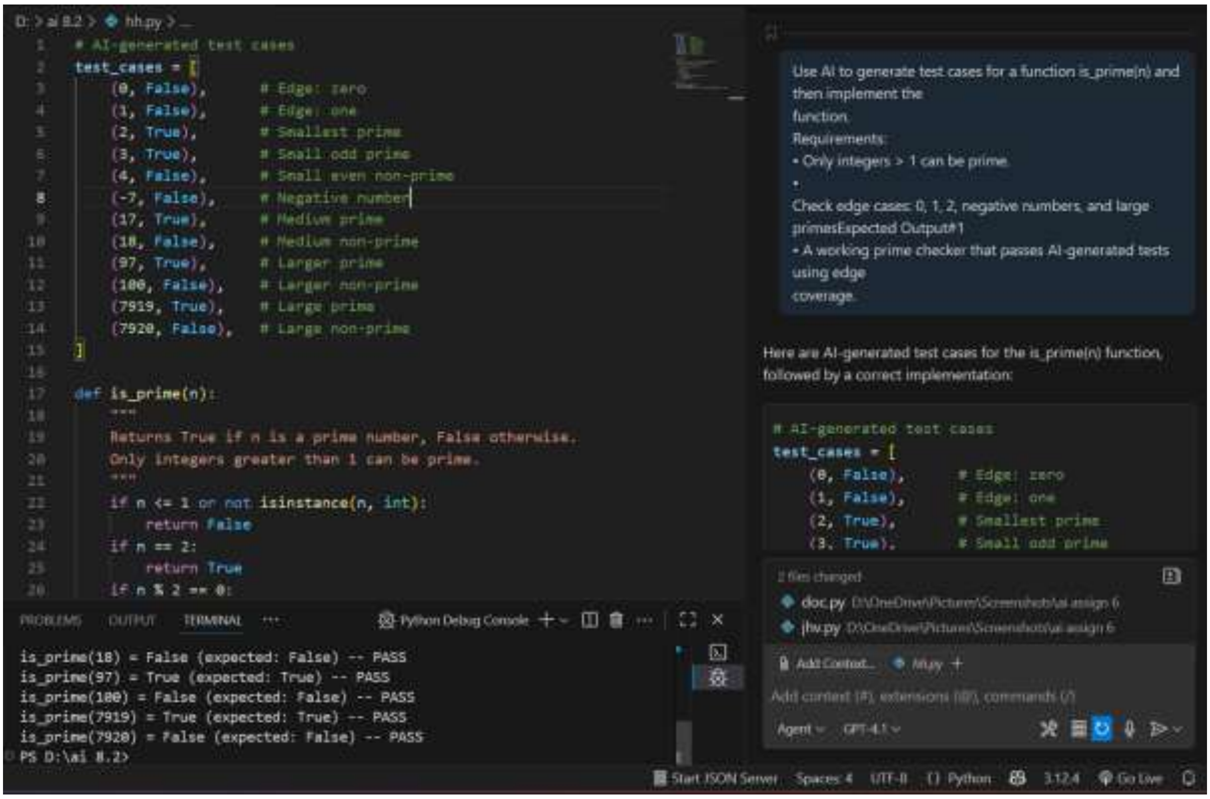
NAME : B.YASHWANTH KUMAR

Roll No : NAME : 2503A51L42

Batch : CSB20

TASK 1

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



D: > ai 8.2 > hh.py > ...

```python
# AI-generated test cases
test_cases = [
    (0, False),       # Edge: zero
    (1, False),       # Edge: one
    (2, True),        # Smallest prime
    (3, True),        # Small odd prime
    (4, False),       # Small even non-prime
    (-7, False),      # Negative number
    (17, True),       # Medium prime
    (18, False),      # Medium non-prime
    (97, True),       # Larger prime
    (100, False),     # Larger non-prime
    (7919, True),     # Large prime
    (7920, False),    # Large non-prime
]

def is_prime(n):
    """
    Returns True if n is a prime number, False otherwise.
    Only integers greater than 1 can be prime.
    """
    if n <= 1 or not isinstance(n, int):
        return False
    if n == 2:
        return True
    if n % 2 == 0:
```

```
is_prime(18) = False (expected: False) -- PASS
is_prime(97) = True (expected: True) -- PASS
is_prime(100) = False (expected: False) -- PASS
is_prime(7919) = True (expected: True) -- PASS
is_prime(7920) = False (expected: False) -- PASS
PS D:\ai 8.2>
```

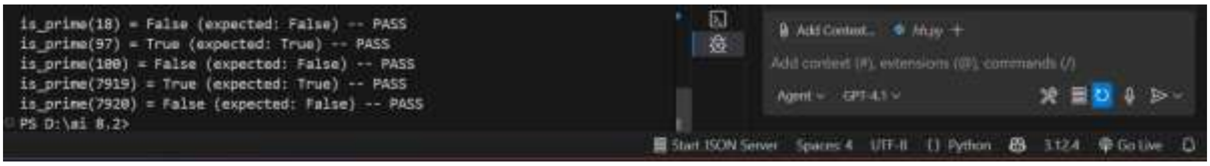
Use AI to generate test cases for a function is_prime(n) and then implement the
function.
Requirements:
• Only integers > 1 can be prime.
•
Check edge cases: 0, 1, 2, negative numbers, and large primesExpected Output#1
• A working prime checker that passes AI-generated tests using edge
coverage.

Here are AI-generated test cases for the is_prime(n) function, followed by a correct implementation:

```python
# AI-generated test cases
test_cases = [
    (0, False),       # Edge: zero
    (1, False),       # Edge: one
    (2, True),        # Smallest prime
    (3, True),        # Small odd prime
```

2 files changed
doc.py D:\OneDrive\Pictures\Screenshots\ai assign 6
jhw.py D:\OneDrive\Pictures\Screenshots\ai assign 6

OUTPUT



```
is_prime(18) = False (expected: False) -- PASS
is_prime(97) = True (expected: True) -- PASS
is_prime(100) = False (expected: False) -- PASS
is_prime(7919) = True (expected: True) -- PASS
is_prime(7920) = False (expected: False) -- PASS
PS D:\ai 8.2>
```

OBSERVATION

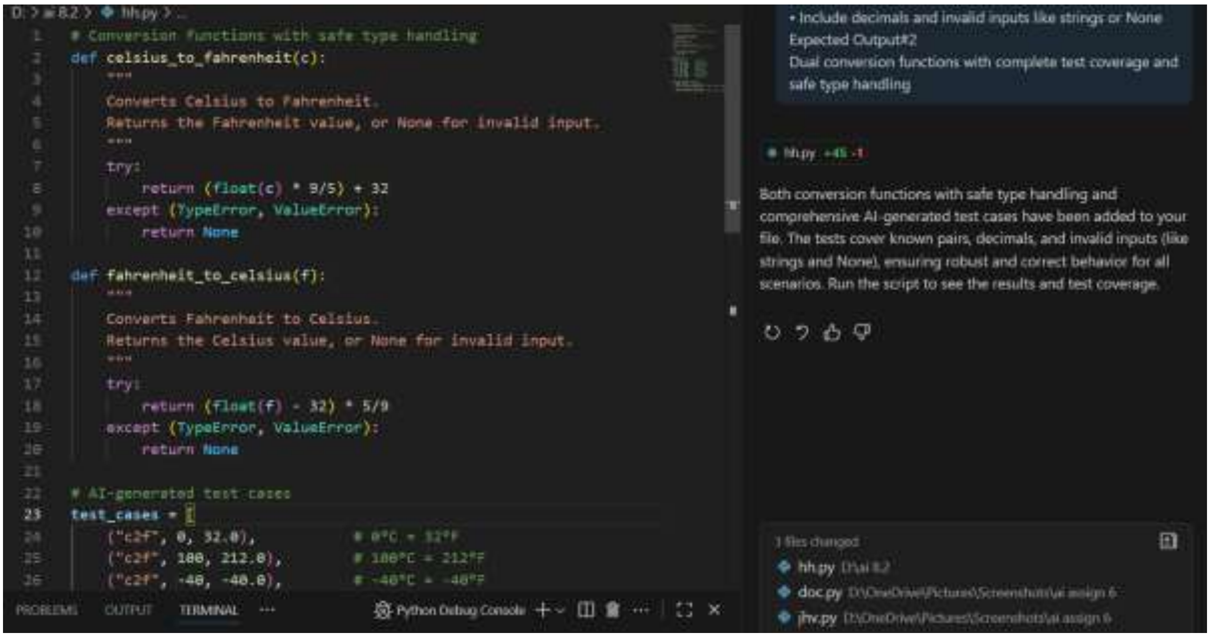
- The is_prime(n) function checks if a number is prime.
    - It returns False for any number less than or equal to 1, or if the input is not an integer.
    - It returns True for 2 (the smallest prime).

  - 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.

TASK 2

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

CODE



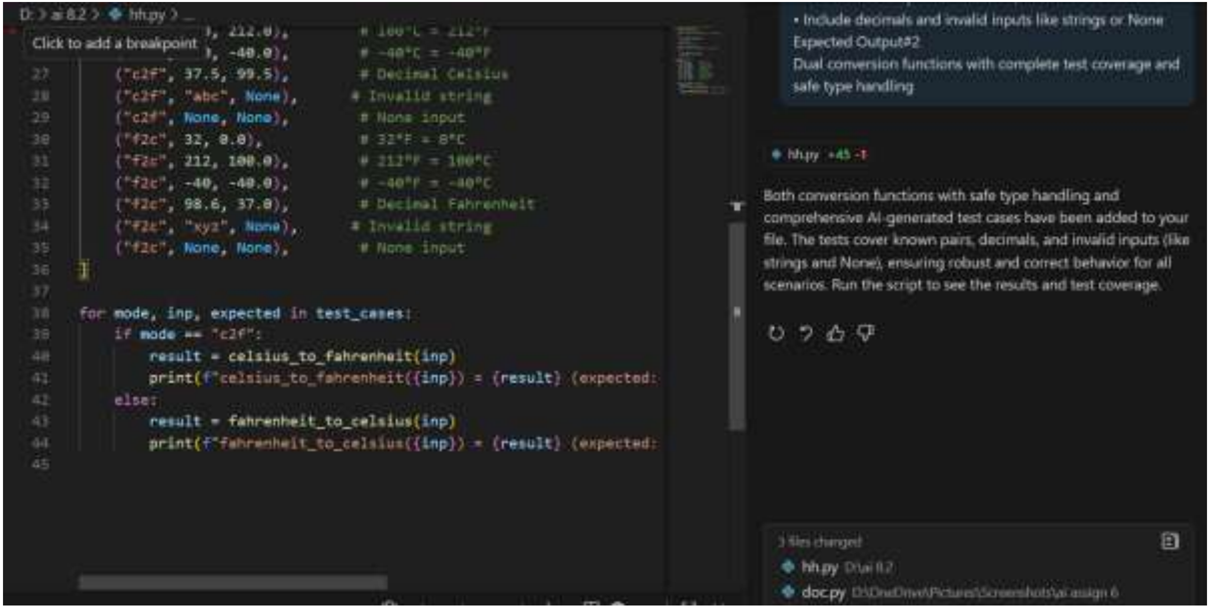
```python
# Conversion functions with safe type handling
def celsius_to_fahrenheit(c):
    """
    Converts Celsius to Fahrenheit.
    Returns the Fahrenheit value, or None for invalid input.
    """
    try:
        return (float(c) * 9/5) + 32
    except (TypeError, ValueError):
        return None

def fahrenheit_to_celsius(f):
    """
    Converts Fahrenheit to Celsius.
    Returns the Celsius value, or None for invalid input.
    """
    try:
        return (float(f) - 32) * 5/9
    except (TypeError, ValueError):
        return None

# AI-generated test cases
test_cases = [
    ("c2f", 0, 32.0),           # 0°C = 32°F
    ("c2f", 100, 212.0),        # 100°C = 212°F
    ("c2f", -40, -40.0),        # -40°C = -40°F
```

• Include decimals and invalid inputs like strings or None
Expected Output#2
Dual conversion functions with complete test coverage and safe type handling

hh.py +45 -1

Both conversion functions with safe type handling and comprehensive AI-generated test cases have been added to your file. The tests cover known pairs, decimals, and invalid inputs (like strings and None), ensuring robust and correct behavior for all scenarios. Run the script to see the results and test coverage.

3 files changed
hh.py D:\ai 8.2
doc.py D:\OneDrive\Pictures\Screenshots\ai assign 6
jhv.py D:\OneDrive\Pictures\Screenshots\ai assign 6



```python
    ("c2f", 37.5, 99.5),        # Decimal Celsius
    ("c2f", "abc", None),      # Invalid string
    ("c2f", None, None),        # None input
    ("f2c", 32, 0.0),           # 32°F = 0°C
    ("f2c", 212, 100.0),        # 212°F = 100°C
    ("f2c", -40, -40.0),        # -40°F = -40°C
    ("f2c", 98.6, 37.0),        # Decimal Fahrenheit
    ("f2c", "xyz", None),      # Invalid string
    ("f2c", None, None),        # None input
]

for mode, inp, expected in test_cases:
    if mode == "c2f":
        result = celsius_to_fahrenheit(inp)
        print(f"celsius_to_fahrenheit({inp}) = {result} (expected:
    else:
        result = fahrenheit_to_celsius(inp)
        print(f"fahrenheit_to_celsius({inp}) = {result} (expected:
```

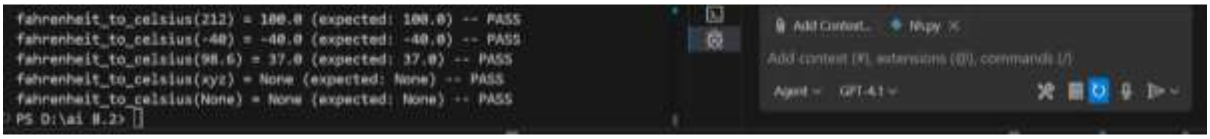
• Include decimals and invalid inputs like strings or None
Expected Output#2
Dual conversion functions with complete test coverage and safe type handling

hh.py +45 -1

Both conversion functions with safe type handling and comprehensive AI-generated test cases have been added to your file. The tests cover known pairs, decimals, and invalid inputs (like strings and None), ensuring robust and correct behavior for all scenarios. Run the script to see the results and test coverage.

3 files changed
hh.py D:\ai 8.2
doc.py D:\OneDrive\Pictures\Screenshots\ai assign 6

OUTPUT



```
fahrenheit_to_celsius(212) = 100.0 (expected: 100.0) -- PASS
fahrenheit_to_celsius(-40) = -40.0 (expected: -40.0) -- PASS
fahrenheit_to_celsius(98.6) = 37.0 (expected: 37.0) -- PASS
fahrenheit_to_celsius(xyz) = None (expected: None) -- PASS
fahrenheit_to_celsius(None) = None (expected: None) -- PASS
PS D:\ai 8.2> 
```

OBSERVATION

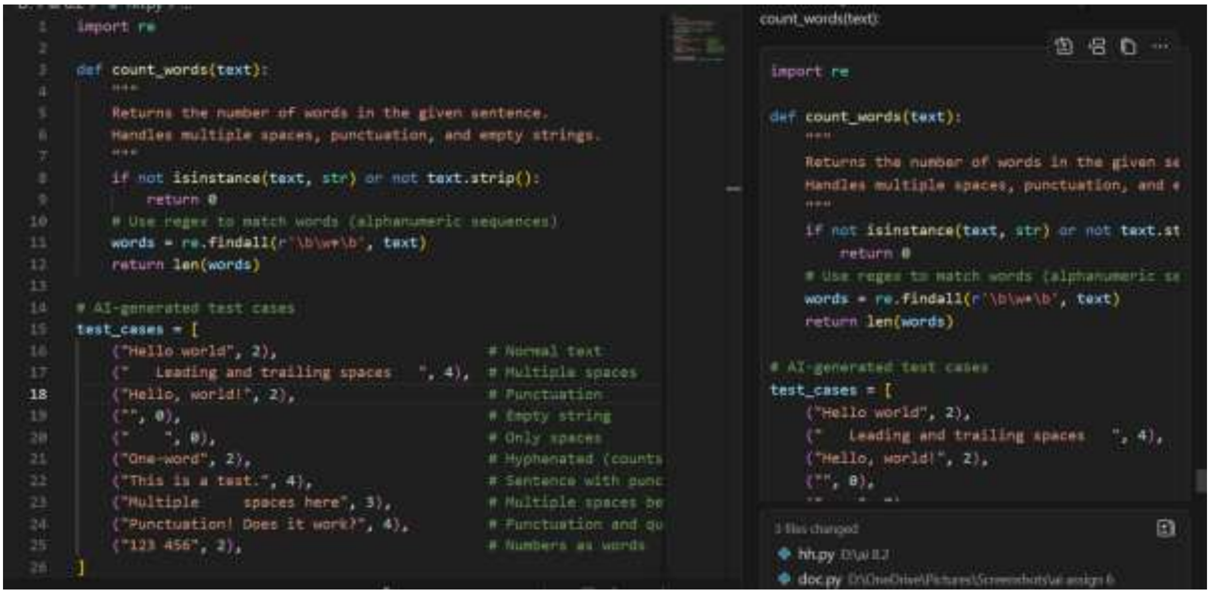
- Two conversion functions are defined:
  - celsius_to_fahrenheit(c): Converts a Celsius value to Fahrenheit. It safely handles invalid input (like strings or None) by returning None if conversion fails.
  - fahrenheit_to_celsius(f): Converts a Fahrenheit value to Celsius, also returning None for invalid 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

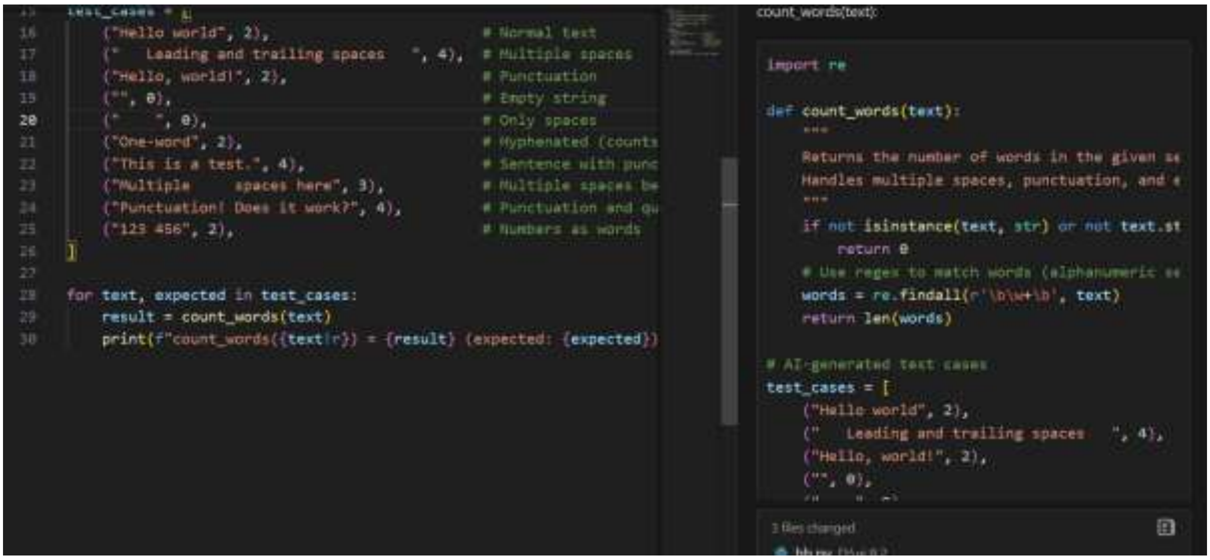
Use AI to write test cases for a function count_words(text) that returns the number of words in a sentence

CODE



```python
import re

def count_words(text):
    """
    Returns the number of words in the given sentence.
    Handles multiple spaces, punctuation, and empty strings.
    """
    if not isinstance(text, str) or not text.strip():
        return 0
    # Use regex to match words (alphanumeric sequences)
    words = re.findall(r'\b\w+\b', text)
    return len(words)

# AI-generated test cases
test_cases = [
    ("Hello world", 2),                          # Normal text
    ("   Leading and trailing spaces   ", 4),    # Multiple spaces
    ("Hello, world!", 2),                        # Punctuation
    ("", 0),                                     # Empty string
    ("    ", 0),                                 # Only spaces
    ("One-word", 2),                             # Hyphenated (counts
    ("This is a test.", 4),                      # Sentence with punc
    ("Multiple     spaces here", 3),             # Multiple spaces be
    ("Punctuation! Does it work?", 4),           # Punctuation and qu
    ("123 456", 2),                              # Numbers as words
]
```

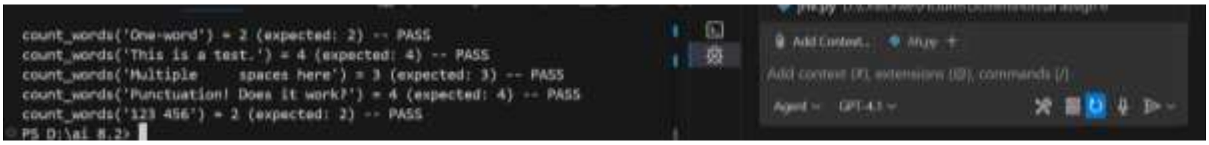
```python
import re

def count_words(text):
    """
    Returns the number of words in the given se
    Handles multiple spaces, punctuation, and e
    """
    if not isinstance(text, str) or not text.st
        return 0
    # Use regex to match words (alphanumeric se
    words = re.findall(r'\b\w+\b', text)
    return len(words)

# AI-generated test cases
test_cases = [
    ("Hello world", 2),
    ("   Leading and trailing spaces   ", 4),
    ("Hello, world!", 2),
    ("", 0),
```



```python
test_cases = [
    ("Hello world", 2),                          # Normal text
    ("   Leading and trailing spaces   ", 4),    # Multiple spaces
    ("Hello, world!", 2),                        # Punctuation
    ("", 0),                                     # Empty string
    ("    ", 0),                                 # Only spaces
    ("One-word", 2),                             # Hyphenated (counts
    ("This is a test.", 4),                      # Sentence with punc
    ("Multiple     spaces here", 3),             # Multiple spaces be
    ("Punctuation! Does it work?", 4),           # Punctuation and qu
    ("123 456", 2),                              # Numbers as words
]

for text, expected in test_cases:
    result = count_words(text)
    print(f"count_words({text!r}) = {result} (expected: {expected})
```

```python
import re

def count_words(text):
    """
    Returns the number of words in the given se
    Handles multiple spaces, punctuation, and e
    """
    if not isinstance(text, str) or not text.st
        return 0
    # Use regex to match words (alphanumeric se
    words = re.findall(r'\b\w+\b', text)
    return len(words)

# AI-generated test cases
test_cases = [
    ("Hello world", 2),
    ("   Leading and trailing spaces   ", 4),
    ("Hello, world!", 2),
    ("", 0),
```

OUTPUT



```
count_words('One-word') = 2 (expected: 2) -- PASS
count_words('This is a test.') = 4 (expected: 4) -- PASS
count_words('Multiple     spaces here') = 3 (expected: 3) -- PASS
count_words('Punctuation! Does it work?') = 4 (expected: 4) -- PASS
count_words('123 456') = 2 (expected: 2) -- PASS
PS D:\ai 8.2>
```

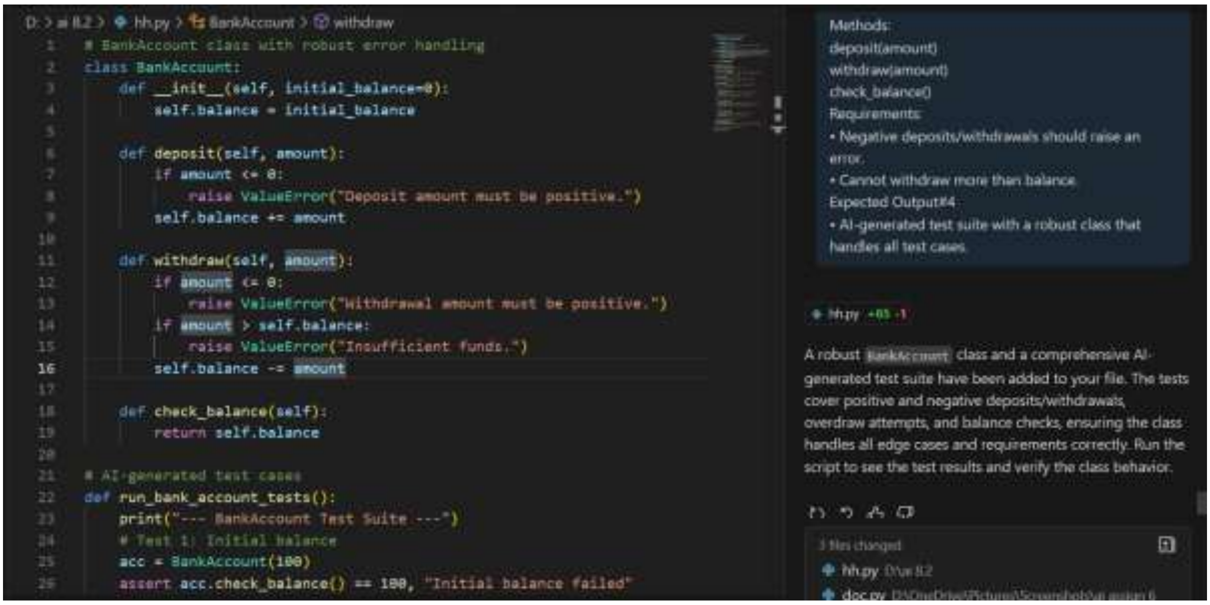
OBSERVATION

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()

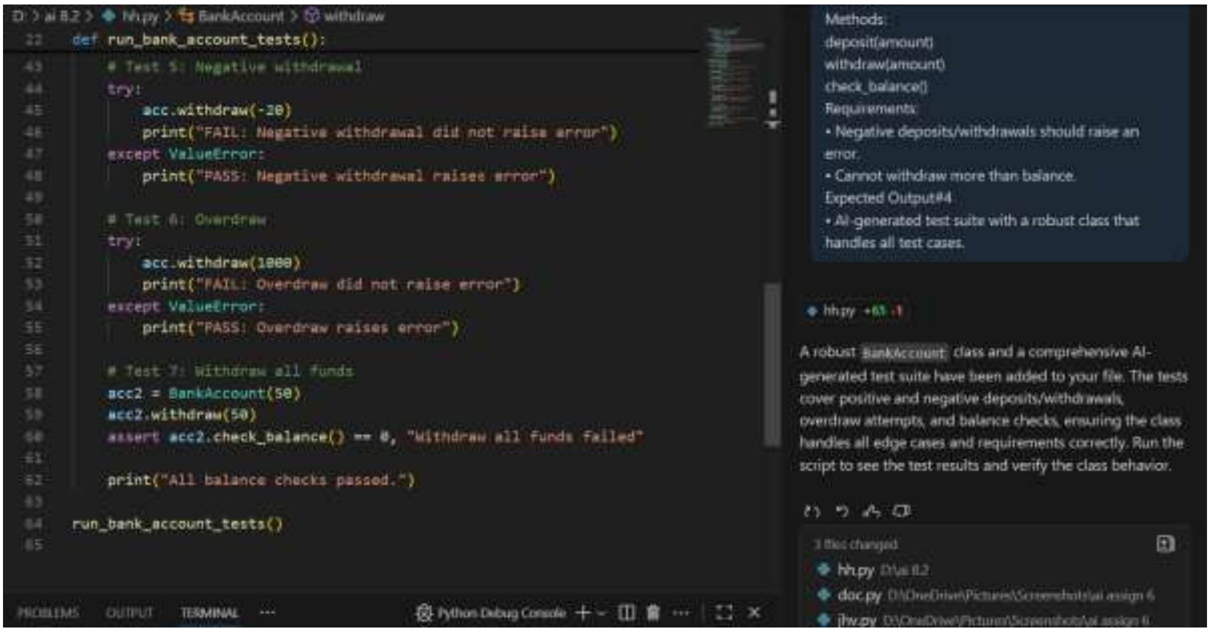
CODE



```python
# BankAccount class with robust error handling
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Deposit amount must be positive.")
        self.balance += amount

    def withdraw(self, amount):
        if amount <= 0:
            raise ValueError("Withdrawal amount must be positive.")
        if amount > self.balance:
            raise ValueError("Insufficient funds.")
        self.balance -= amount

    def check_balance(self):
        return self.balance

# AI-generated test cases
def run_bank_account_tests():
    print("--- BankAccount Test Suite ---")
    # Test 1: Initial balance
    acc = BankAccount(100)
    assert acc.check_balance() == 100, "Initial balance failed"
```

Methods:
deposit(amount)
withdraw(amount)
check_balance()
Requirements:
• Negative deposits/withdrawals should raise an error.
• Cannot withdraw more than balance.
Expected Output#4
• AI-generated test suite with a robust class that handles all test cases.

A robust BankAccount class and a comprehensive AI-generated test suite have been added to your file. The tests cover positive and negative deposits/withdrawals, overdraw attempts, and balance checks, ensuring the class handles all edge cases and requirements correctly. Run the script to see the test results and verify the class behavior.



```python
def run_bank_account_tests():
    # Test 5: Negative withdrawal
    try:
        acc.withdraw(-20)
        print("FAIL: Negative withdrawal did not raise error")
    except ValueError:
        print("PASS: Negative withdrawal raisee error")

    # Test 6: Overdraw
    try:
        acc.withdraw(1000)
        print("FAIL: Overdraw did not raise error")
    except ValueError:
        print("PASS: Overdraw raises error")

    # Test 7: Withdraw all funds
    acc2 = BankAccount(50)
    acc2.withdraw(50)
    assert acc2.check_balance() == 0, "Withdraw all funds failed"

    print("All balance checks passed.")

run_bank_account_tests()
```

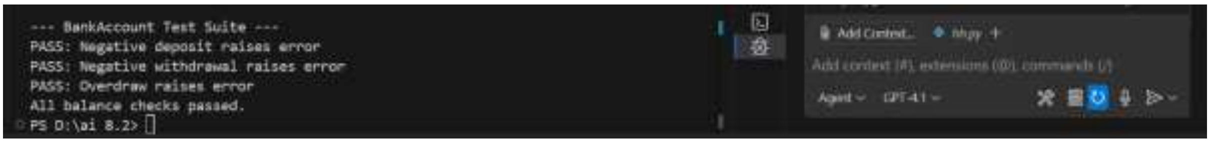
Methods:
deposit(amount)
withdraw(amount)
check_balance()
Requirements:
• Negative deposits/withdrawals should raise an error.
• Cannot withdraw more than balance.
Expected Output#4
• AI-generated test suite with a robust class that handles all test cases.

hh.py +65 -1

A robust `BankAccount` class and a comprehensive AI-generated test suite have been added to your file. The tests cover positive and negative deposits/withdrawals, overdraw attempts, and balance checks, ensuring the class handles all edge cases and requirements correctly. Run the script to see the test results and verify the class behavior.

3 files changed
hh.py D:\ai 8.2
doc.py D:\OneDrive\Pictures\Screenshots\ai assign 6
jhw.py D:\OneDrive\Pictures\Screenshots\ai assign 6

OUTPUT



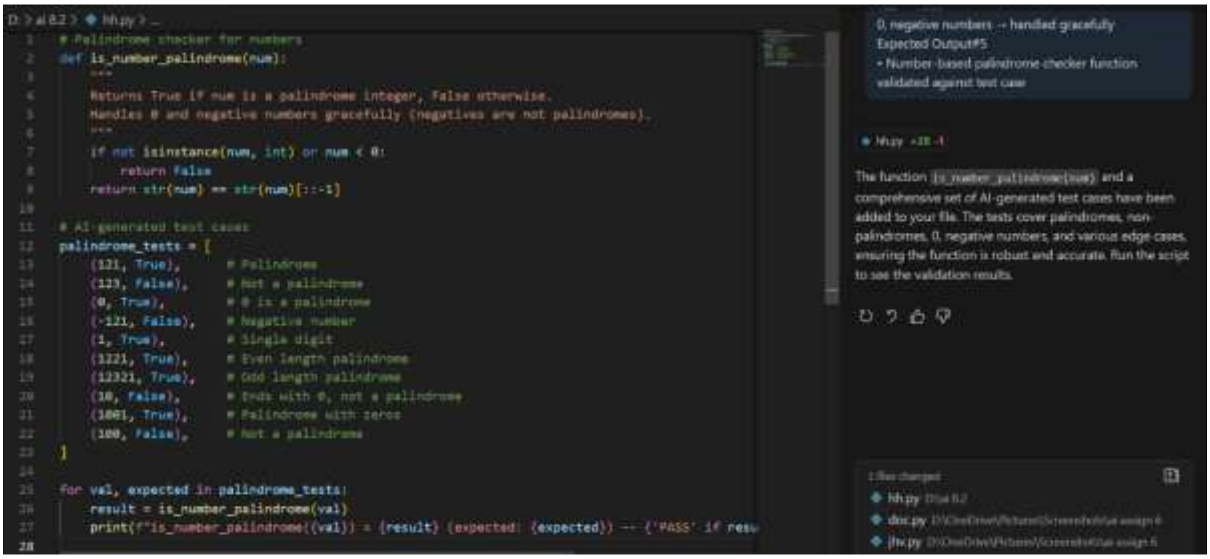
```
--- BankAccount Test Suite ---
PASS: Negative deposit raises error
PASS: Negative withdrawal raises error
PASS: Overdraw raises error
All balance checks passed.
PS D:\ai 8.2> 
```

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



D: > ai 8.2 > hh.py > ...

```python
# Palindrome checker for numbers
def is_number_palindrome(num):
    """
    Returns True if num is a palindrome integer, False otherwise.
    Handles 0 and negative numbers gracefully (negatives are not palindromes).
    """
    if not isinstance(num, int) or num < 0:
        return False
    return str(num) == str(num)[::-1]

# AI-generated test cases
palindrome_tests = [
    (121, True),       # Palindrome
    (123, False),      # Not a palindrome
    (0, True),         # 0 is a palindrome
    (-121, False),     # Negative number
    (1, True),         # Single digit
    (1221, True),      # Even length palindrome
    (12321, True),     # Odd length palindrome
    (10, False),       # Ends with 0, not a palindrome
    (1001, True),      # Palindrome with zeros
    (100, False),      # Not a palindrome
]

for val, expected in palindrome_tests:
    result = is_number_palindrome(val)
    print(f"is_number_palindrome({val}) = {result} (expected: {expected}) -- {'PASS' if resu
```

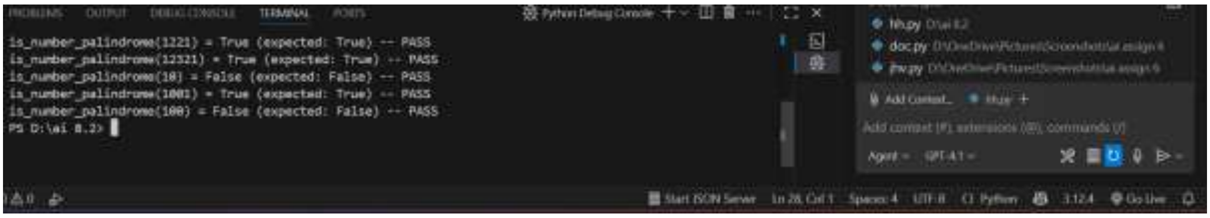
0, negative numbers → handled gracefully
Expected Output#5
• Number-based palindrome checker function validated against test case

hh.py +28 -1

The function `is_number_palindrome(num)` and a comprehensive set of AI-generated test cases have been added to your file. The tests cover palindromes, non-palindromes, 0, negative numbers, and various edge cases, ensuring the function is robust and accurate. Run the script to see the validation results.

1 files changed
- hh.py D:\ai 8.2
- doc.py D:\OneDrive\Pictures\Screenshots\ai assign 6
- jhv.py D:\OneDrive\Pictures\Screenshots\ai assign 6

OUTPUT



```
is_number_palindrome(1221) = True (expected: True) -- PASS
is_number_palindrome(12321) = True (expected: True) -- PASS
is_number_palindrome(10) = False (expected: False) -- PASS
is_number_palindrome(1001) = True (expected: True) -- PASS
is_number_palindrome(100) = False (expected: False) -- PASS
PS D:\ai 8.2> 
```

- hh.py D:\ai 8.2
- doc.py D:\OneDrive\Pictures\Screenshots\ai assign 6
- jhv.py D:\OneDrive\Pictures\Screenshots\ai assign 6

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.