

NAME :M.Ramsai

Roll No : 2503A51L53

Batch : CSB20

TASK 1

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

The screenshot shows a code editor with a Python file named `hh.py`. The code includes a list of test cases and a function `is_prime(n)`. The test cases are: `(0, False)`, `(1, False)`, `(2, True)`, `(3, True)`, `(4, False)`, `(-7, False)`, `(17, True)`, `(18, False)`, `(97, True)`, `(100, False)`, `(7919, True)`, and `(7920, False)`. The function `is_prime(n)` checks if `n` is a prime number. It returns `False` for `n <= 1` or if `n` is not an integer. It returns `True` for `n == 2` and `False` for even numbers greater than 2. The terminal output shows that all test cases passed.

```
D: > ai 8.2 > hh.py > ...
1 # AI-generated test cases
2 test_cases = [
3     (0, False), # Edge: zero
4     (1, False), # Edge: one
5     (2, True),  # Smallest prime
6     (3, True),  # Small odd prime
7     (4, False), # Small even non-prime
8     (-7, False), # Negative number
9     (17, True),  # Medium prime
10    (18, False),  # Medium non-prime
11    (97, True),   # Larger prime
12    (100, False), # Larger non-prime
13    (7919, True), # Large prime
14    (7920, False), # Large non-prime
15 ]
16
17 def is_prime(n):
18     """
19     Returns True if n is a prime number, False otherwise.
20     Only integers greater than 1 can be prime.
21     """
22     if n <= 1 or not isinstance(n, int):
23         return False
24     if n == 2:
25         return True
26     if n % 2 == 0:
27         return False
28     for i in range(3, int(n**0.5) + 1, 2):
29         if n % i == 0:
30             return False
31     return True
32
33 # Test cases
34 for n, expected in test_cases:
35     result = is_prime(n)
36     print(f"is_prime({n}) = {result} (expected: {expected}) -- {'PASS' if result == expected else 'FAIL'}")
37
38 is_prime(18) = False (expected: False) -- PASS
39 is_prime(97) = True (expected: True) -- PASS
40 is_prime(100) = False (expected: False) -- PASS
41 is_prime(7919) = True (expected: True) -- PASS
42 is_prime(7920) = False (expected: False) -- PASS
43 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:

```
# 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
- `jhv.py` D:\OneDrive\Pictures\Screenshots\ai assign 6

Add Context... `hh.py` +

Add context (#), extensions (@), commands (/)

Agent GPT-4.1

OUTPUT

The screenshot shows the terminal output of the test cases. All test cases passed.

```
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

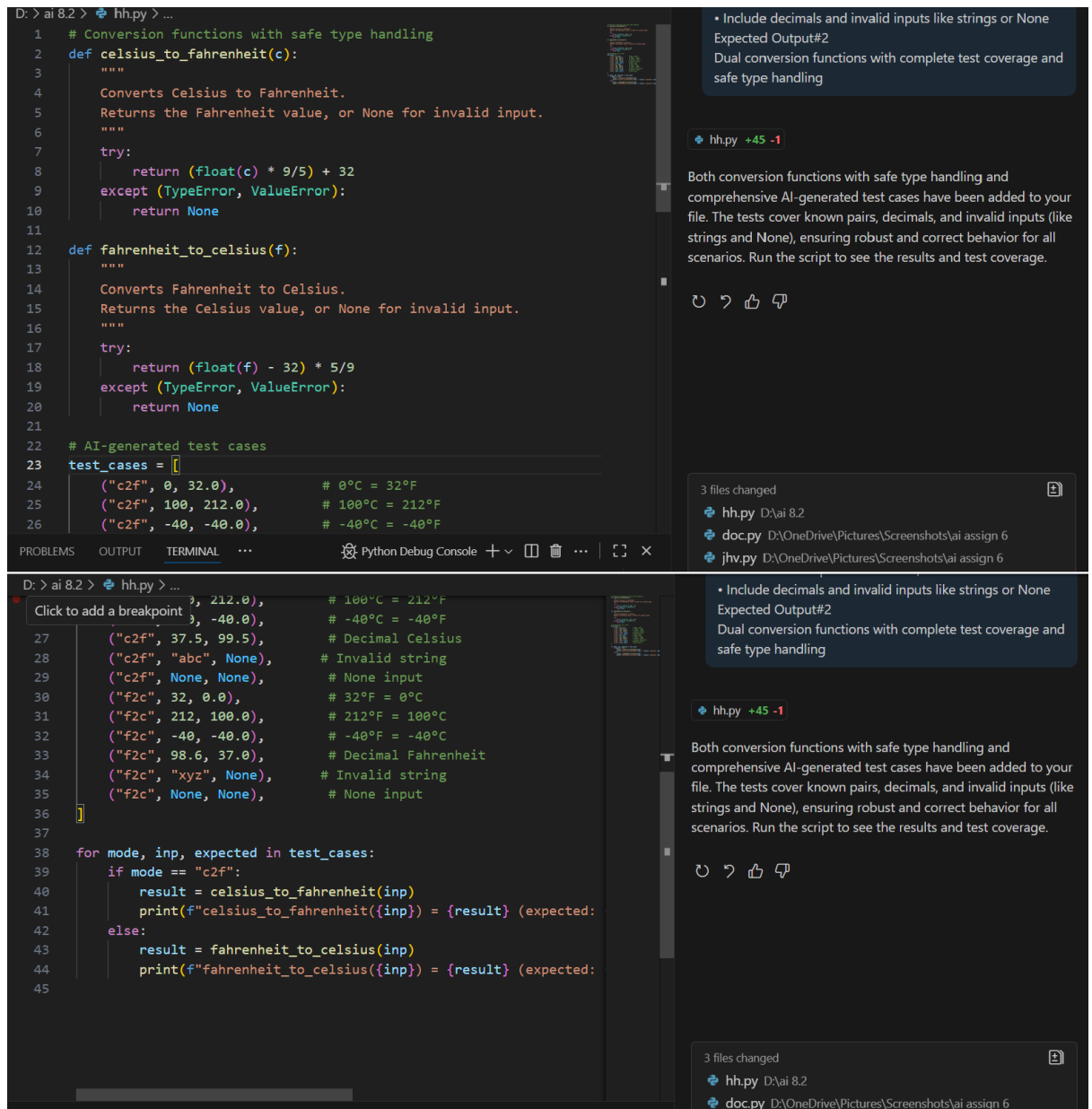
- The `is_prime(n)` function checks if a number is prime.
 - It returns `False` for any number less than or equal to 1, or if the input is not an integer.
 - It returns `True` for 2 (the smallest prime).

- 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



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., $0^{\circ}\text{C} = 32^{\circ}\text{F}$, $100^{\circ}\text{C} = 212^{\circ}\text{F}$, $-40^{\circ}\text{C} = -40^{\circ}\text{F}$).
 - 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

```
1 import re
2
3 def count_words(text):
4     """
5     Returns the number of words in the given sentence.
6     Handles multiple spaces, punctuation, and empty strings.
7     """
8     if not isinstance(text, str) or not text.strip():
9         return 0
10    # Use regex to match words (alphanumeric sequences)
11    words = re.findall(r'\b\w+\b', text)
12    return len(words)
13
14 # AI-generated test cases
15 test_cases = [
16     ("Hello world", 2), # Normal text
17     ("  Leading and trailing spaces  ", 4), # Multiple spaces
18     ("Hello, world!", 2), # Punctuation
19     ("", 0), # Empty string
20     (" ", 0), # Only spaces
21     ("One-word", 2), # Hyphenated (counts
22     ("This is a test.", 4), # Sentence with punc
23     ("Multiple   spaces here", 3), # Multiple spaces be
24     ("Punctuation! Does it work?", 4), # Punctuation and qu
25     ("123 456", 2), # Numbers as words
26 ]
27
28 for text, expected in test_cases:
29     result = count_words(text)
30     print(f"count_words('{text!r}') = {result} (expected: {expected})")
```

```
count_words(text):
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),
    (" ", 0),
    ("One-word", 2),
    ("This is a test.", 4),
    ("Multiple   spaces here", 3),
    ("Punctuation! Does it work?", 4),
    ("123 456", 2),
]
```

```
count_words(text):
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),
    (" ", 0),
    ("One-word", 2),
    ("This is a test.", 4),
    ("Multiple   spaces here", 3),
    ("Punctuation! Does it work?", 4),
    ("123 456", 2),
]
```

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
```

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

```
D:\> ai 8.2 > hh.py > BankAccount > withdraw
1 # BankAccount class with robust error handling
2 class BankAccount:
3     def __init__(self, initial_balance=0):
4         self.balance = initial_balance
5
6     def deposit(self, amount):
7         if amount <= 0:
8             raise ValueError("Deposit amount must be positive.")
9         self.balance += amount
10
11     def withdraw(self, amount):
12         if amount <= 0:
13             raise ValueError("Withdrawal amount must be positive.")
14         if amount > self.balance:
15             raise ValueError("Insufficient funds.")
16         self.balance -= amount
17
18     def check_balance(self):
19         return self.balance
20
21 # AI-generated test cases
22 def run_bank_account_tests():
23     print("--- BankAccount Test Suite ---")
24     # Test 1: Initial balance
25     acc = BankAccount(100)
26     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.

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_assignment 6

```
D: > ai 8.2 > hh.py > BankAccount > withdraw
22 def run_bank_account_tests():
43     # Test 5: Negative withdrawal
44     try:
45         acc.withdraw(-20)
46         print("FAIL: Negative withdrawal did not raise error")
47     except ValueError:
48         print("PASS: Negative withdrawal raises error")
49
50     # Test 6: Overdraw
51     try:
52         acc.withdraw(1000)
53         print("FAIL: Overdraw did not raise error")
54     except ValueError:
55         print("PASS: Overdraw raises error")
56
57     # Test 7: Withdraw all funds
58     acc2 = BankAccount(50)
59     acc2.withdraw(50)
60     assert acc2.check_balance() == 0, "Withdraw all funds failed"
61
62     print("All balance checks passed.")
63
64     run_bank_account_tests()
65
```

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
• jhv.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>
```

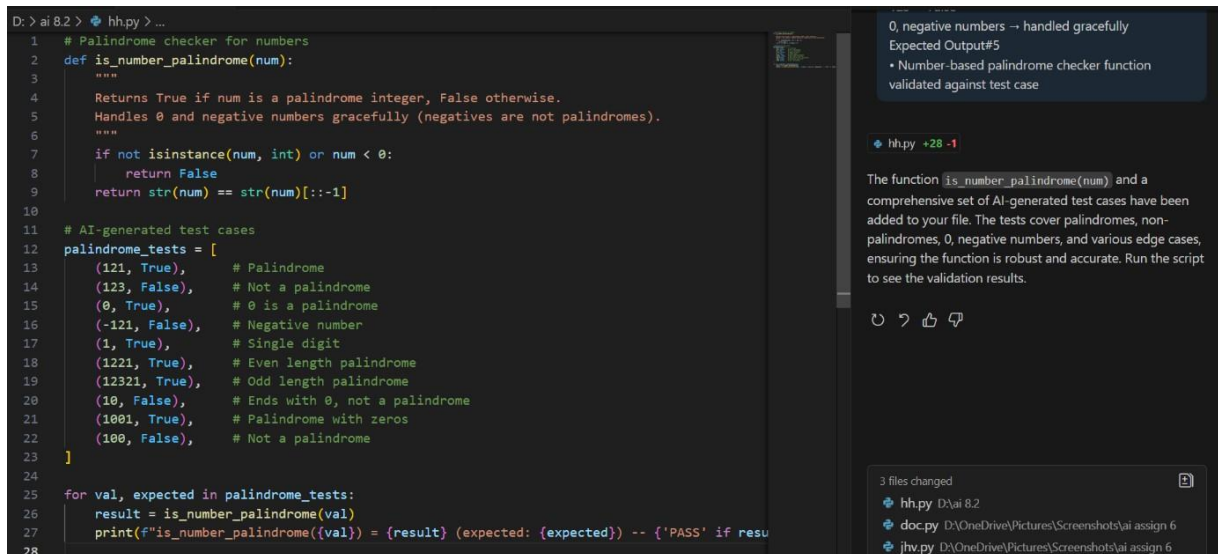
Add Context... hh.py +
Add context (#), extensions (@), commands (/)
Agent GPT-4.1

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 ...
1 # Palindrome checker for numbers
2 def is_number_palindrome(num):
3     """
4     Returns True if num is a palindrome integer, False otherwise.
5     Handles 0 and negative numbers gracefully (negatives are not palindromes).
6     """
7     if not isinstance(num, int) or num < 0:
8         return False
9     return str(num) == str(num)[::-1]
10
11 # AI-generated test cases
12 palindrome_tests = [
13     (121, True),      # Palindrome
14     (123, False),     # Not a palindrome
15     (0, True),        # 0 is a palindrome
16     (-121, False),    # Negative number
17     (1, True),        # Single digit
18     (1221, True),     # Even length palindrome
19     (12321, True),    # Odd length palindrome
20     (10, False),      # Ends with 0, not a palindrome
21     (1001, True),     # Palindrome with zeros
22     (100, False),     # Not a palindrome
23 ]
24
25 for val, expected in palindrome_tests:
26     result = is_number_palindrome(val)
27     print(f"is_number_palindrome({val}) = {result} (expected: {expected}) -- {'PASS' if result == expected else 'FAIL'}")
28
```

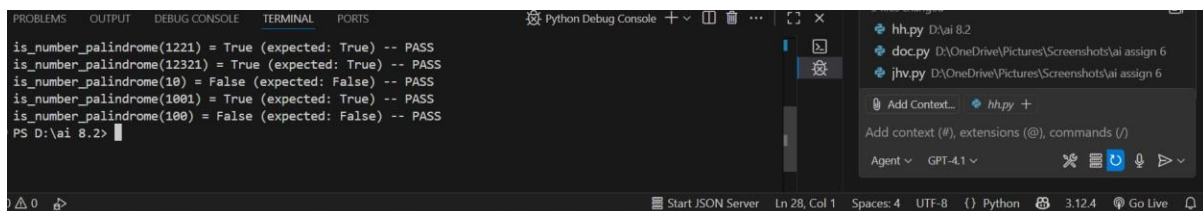
0, negative numbers → handled gracefully
Expected Output#5
• Number-based palindrome checker function validated against test case

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.

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

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>
```

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.