

# SR UNIVERSITY

## AI ASSISTED CODING

**NAME:**BEGALA HASINI

**HT.NO:**2503A51L13

### Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases

#### Lab Objectives:

- To introduce students to test-driven development (TDD) using AI code generation tools.
- To enable the generation of test cases before writing code implementations.
- To reinforce the importance of testing, validation, and error handling.
- To encourage writing clean and reliable code based on AI-generated test expectations.

#### Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Use AI tools to write test cases for Python functions and classes.
- Implement functions based on test cases in a test-first development style.
- Use unittest or pytest to validate code correctness.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic

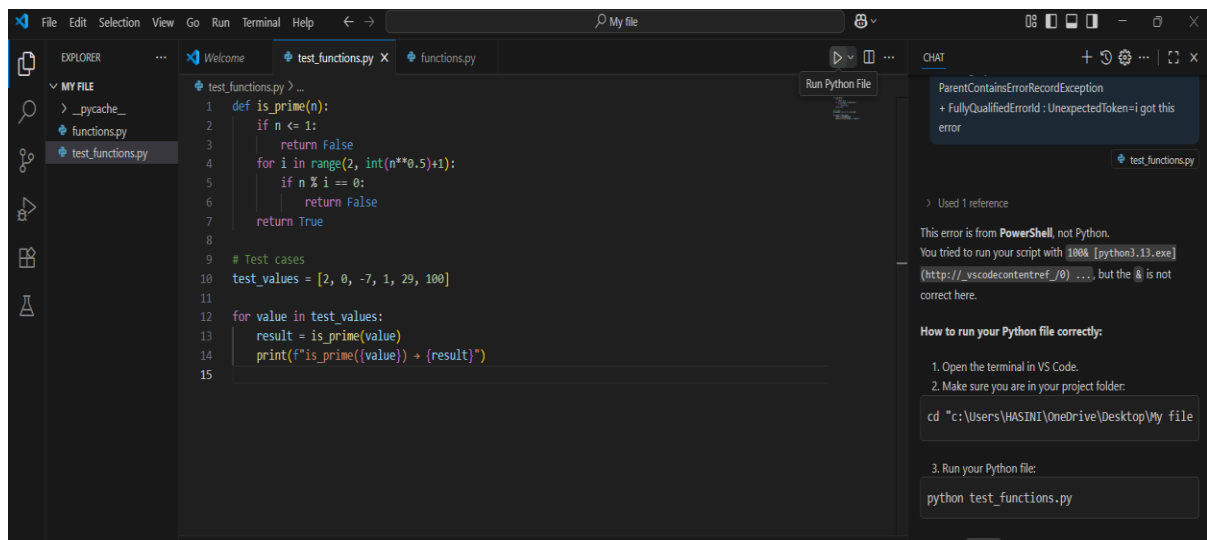
#### Task Description#1

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

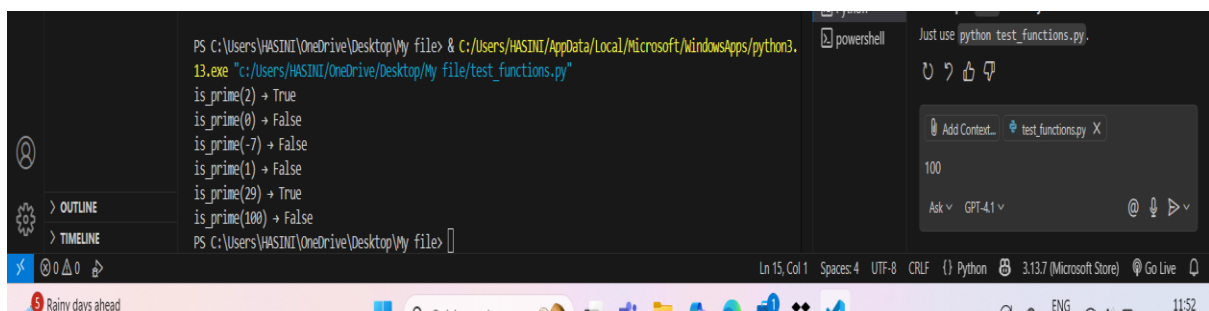
Requirements:

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

**CODE:**



## OUTPUT:



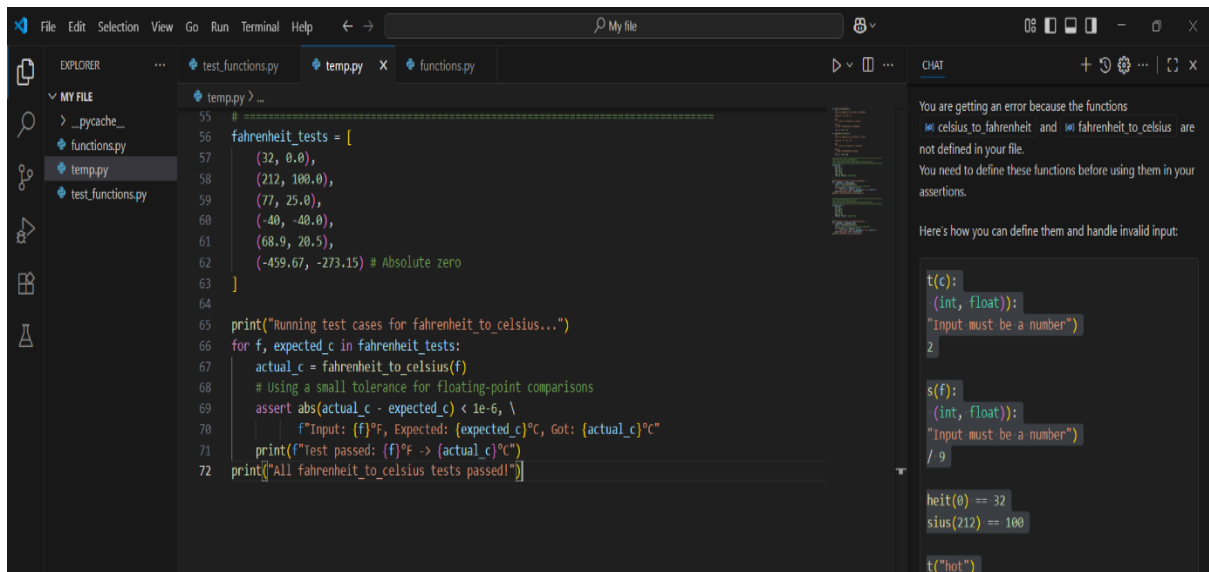
## Task2#Description(Loops)

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

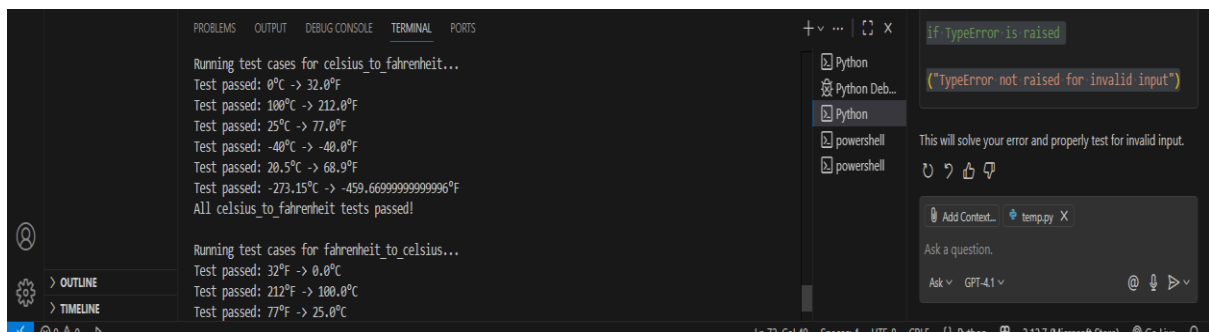
## Requirements

- Validate known pairs:  $0^{\circ}\text{C} = 32^{\circ}\text{F}$ ,  $100^{\circ}\text{C} = 212^{\circ}\text{F}$ .
- Include decimals and invalid inputs like strings or none.

## CODE:



## OUTPUT:



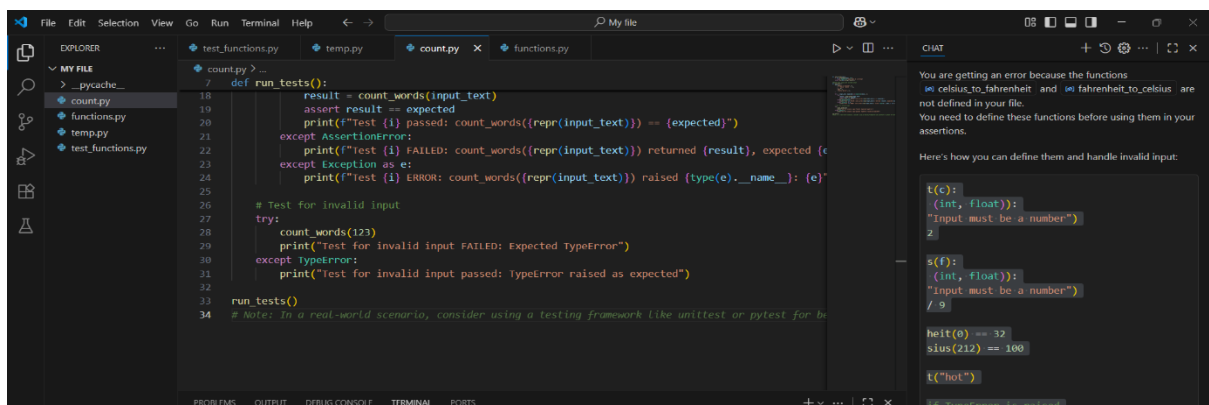
## TaskDescription#3

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

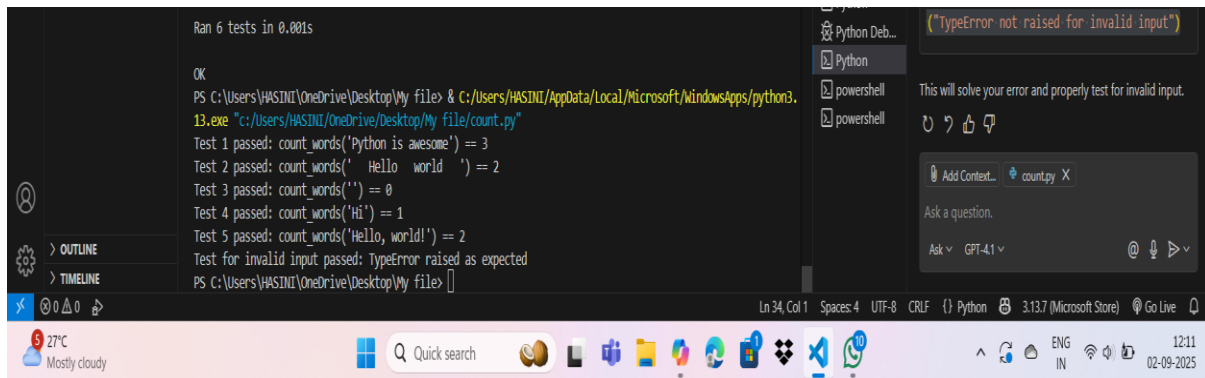
## Requirement

Handle normal text, multiple spaces, punctuation, and empty and strings.

## CODE:



## OUTPUT:



```
Ran 6 tests in 0.001s

OK
PS C:\Users\HASINI\OneDrive\Desktop\My file> & C:/Users/HASINI/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/HASINI/OneDrive/Desktop/My file/count.py"
Test 1 passed: count_words('Python is awesome') == 3
Test 2 passed: count_words(' Hello world ') == 2
Test 3 passed: count_words('') == 0
Test 4 passed: count_words('Hi') == 1
Test 5 passed: count_words('Hello, world!') == 2
Test for invalid input passed: TypeError raised as expected
PS C:\Users\HASINI\OneDrive\Desktop\My file> ]
```

## Task Description#4

- Generate test cases for a **BankAccount** class with:

### Methods:

**deposit(amount)**

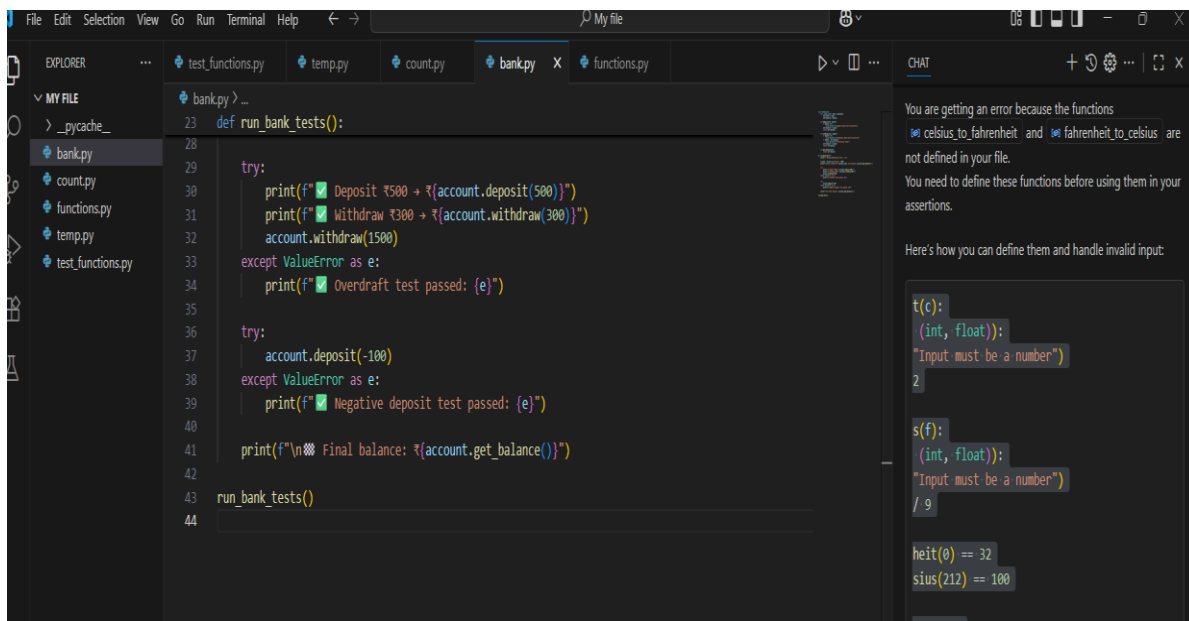
**withdraw(amount)**

**check\_balance()**

### Requirements:

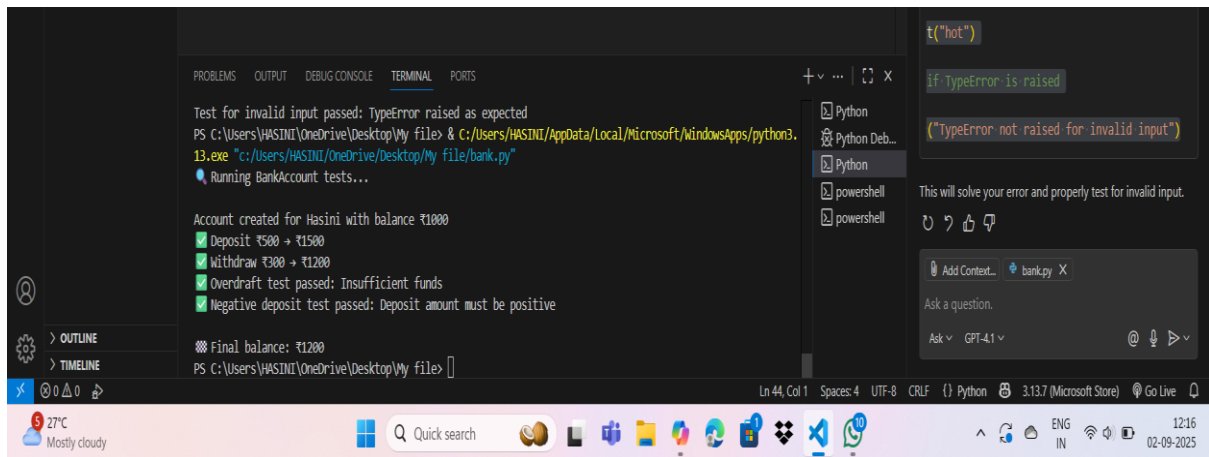
- Negative deposits/withdrawals should raise an error.
- Cannot withdraw more than balance.

## CODE:



```
File Edit Selection View Go Run Terminal Help
EXPLORER
  MY FILE
    _pycache_
    bank.py
    count.py
    functions.py
    temp.py
    test_functions.py
  bank.py
23 def run_bank_tests():
24
25     try:
26         print(f"Deposit ₹500 → ₹{account.deposit(500)}")
27         print(f"Withdraw ₹300 → ₹{account.withdraw(300)}")
28         account.withdraw(1500)
29     except ValueError as e:
30         print(f"Overdraft test passed: {e}")
31
32     try:
33         account.deposit(-100)
34     except ValueError as e:
35         print(f"Negative deposit test passed: {e}")
36
37     print(f"Final balance: ₹{account.get_balance()}")
38
39 run_bank_tests()
40
```

## OUTPUT:



## Task Description#5

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

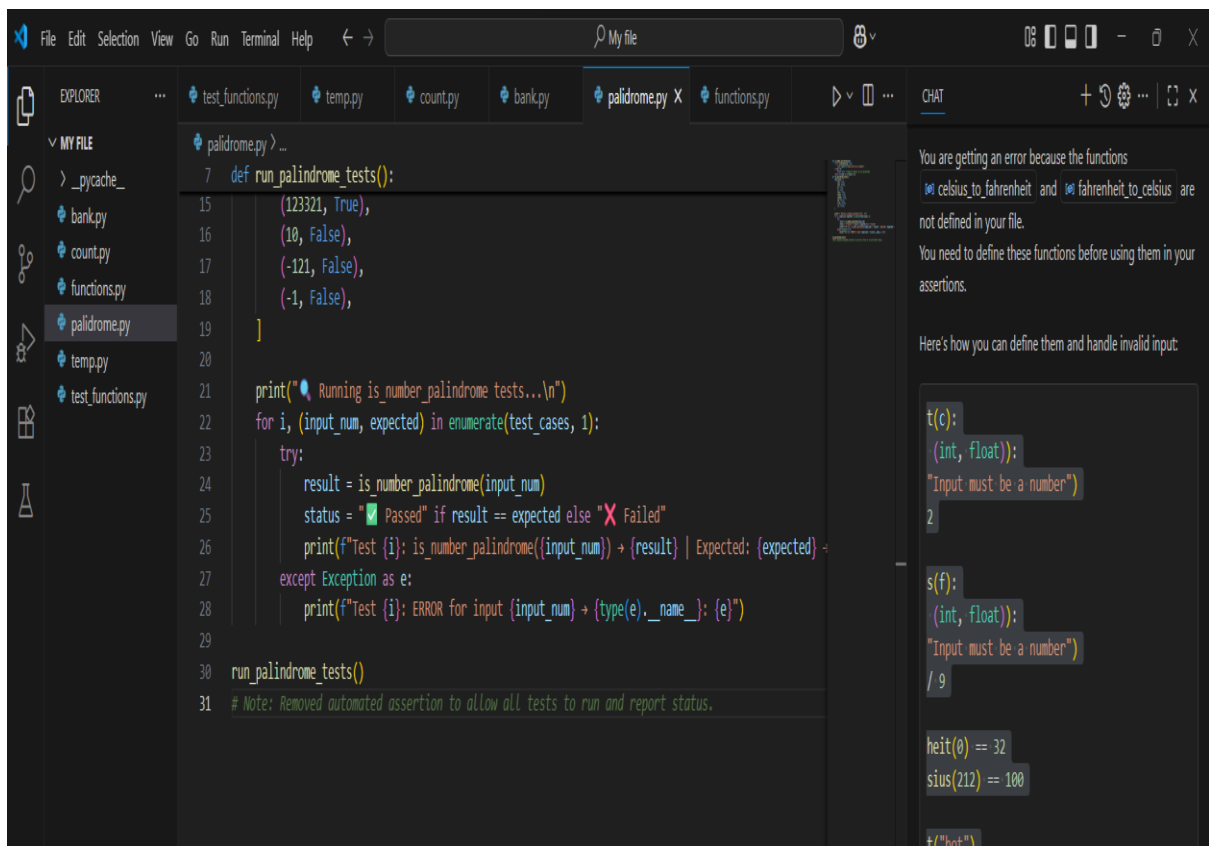
Examples:

121 → True

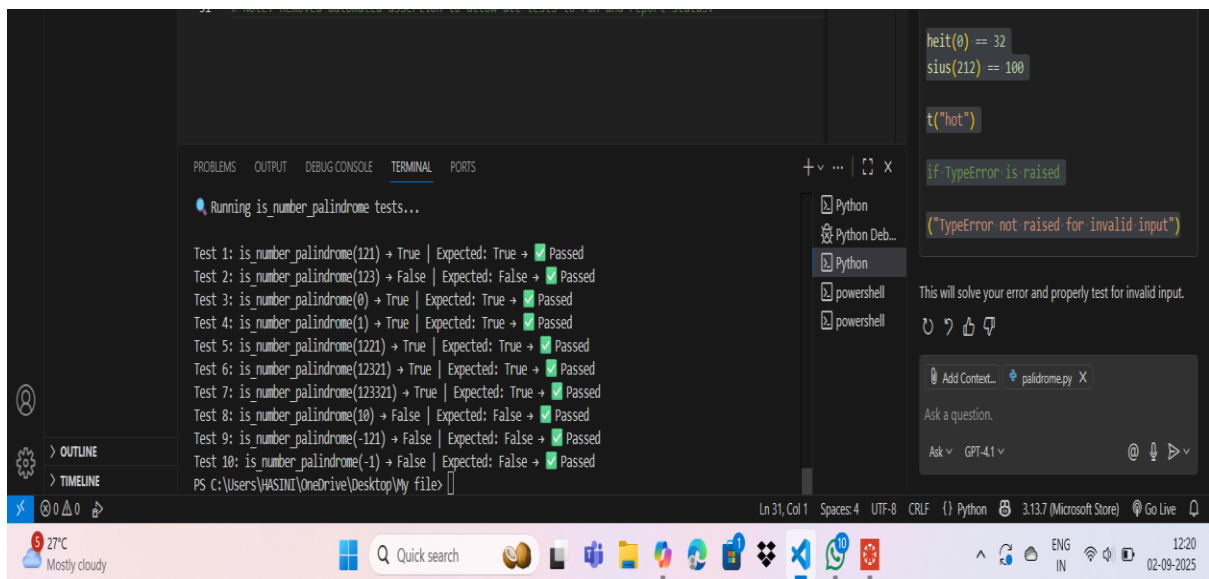
123 → False

0, negative numbers → handled gracefully.

CODE:



## OUTPUT:



The screenshot shows a VS Code editor with a terminal window displaying the results of running tests for a function named `is_number_palindrome`. The tests are as follows:

Test	Input	Expected	Actual	Result
Test 1	<code>is_number_palindrome(121)</code>	<code>True</code>	<code>True</code>	Passed
Test 2	<code>is_number_palindrome(123)</code>	<code>False</code>	<code>False</code>	Passed
Test 3	<code>is_number_palindrome(0)</code>	<code>True</code>	<code>True</code>	Passed
Test 4	<code>is_number_palindrome(1)</code>	<code>True</code>	<code>True</code>	Passed
Test 5	<code>is_number_palindrome(1221)</code>	<code>True</code>	<code>True</code>	Passed
Test 6	<code>is_number_palindrome(12321)</code>	<code>True</code>	<code>True</code>	Passed
Test 7	<code>is_number_palindrome(123321)</code>	<code>True</code>	<code>True</code>	Passed
Test 8	<code>is_number_palindrome(10)</code>	<code>False</code>	<code>False</code>	Passed
Test 9	<code>is_number_palindrome(-121)</code>	<code>False</code>	<code>False</code>	Passed
Test 10	<code>is_number_palindrome(-1)</code>	<code>False</code>	<code>False</code>	Passed

The terminal also shows the command `PS C:\Users\HASINI\OneDrive\Desktop\My file>` and the output of the tests: `Running is_number_palindrome tests...`

On the right side of the editor, there is a snippet of Python code:

```
heit(0) == 32
sius(212) == 100

t("hot")

if TypeError is raised
("TypeError not raised for invalid input")
```

Below the code, there is a message: "This will solve your error and properly test for invalid input." and a button to "Add Context..." with a file named `palindrome.py`.

## OBSERVATIONS:

By completing this assignment, I am able to:

- Use AI tools to write test cases for Python functions and classes.
- Implement functions based on test cases in a test-first development style.
- Use unittest or pytest to validate code correctness.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic.