ASSIGNMENT-7.1

AI

BATCH: 18

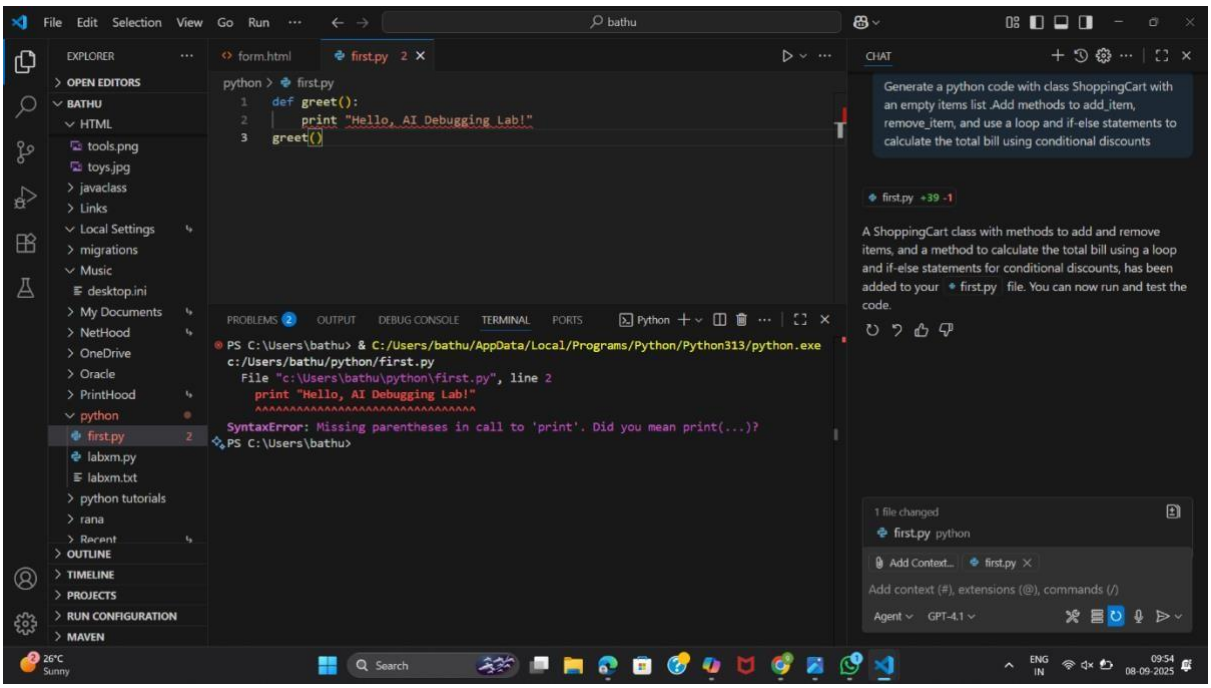
2403A52187

Task Description 1:-

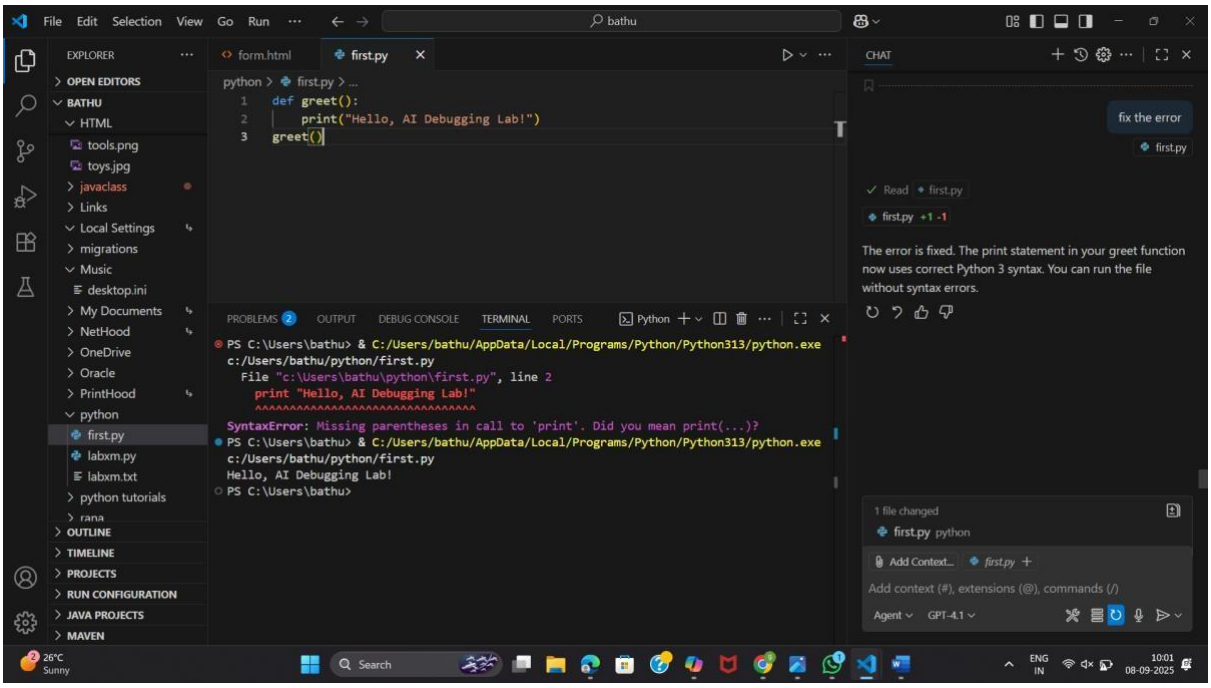
(Syntax Errors – Missing Parentheses in Print
Statement)
Task: Provide a Python snippet with a missing parenthesis in a print
statement (e.g., print "Hello"). Use AI to detect and fix the syntax error.
Bug: Missing parentheses in print statement def
greet():
print "Hello, AI Debugging Lab!"
greet()
Requirements:
- Run the given code to observe the error.
- Apply AI suggestions to correct the syntax.
- Use at least 3 assert test cases to confirm the corrected code works.

Expected Output #1:

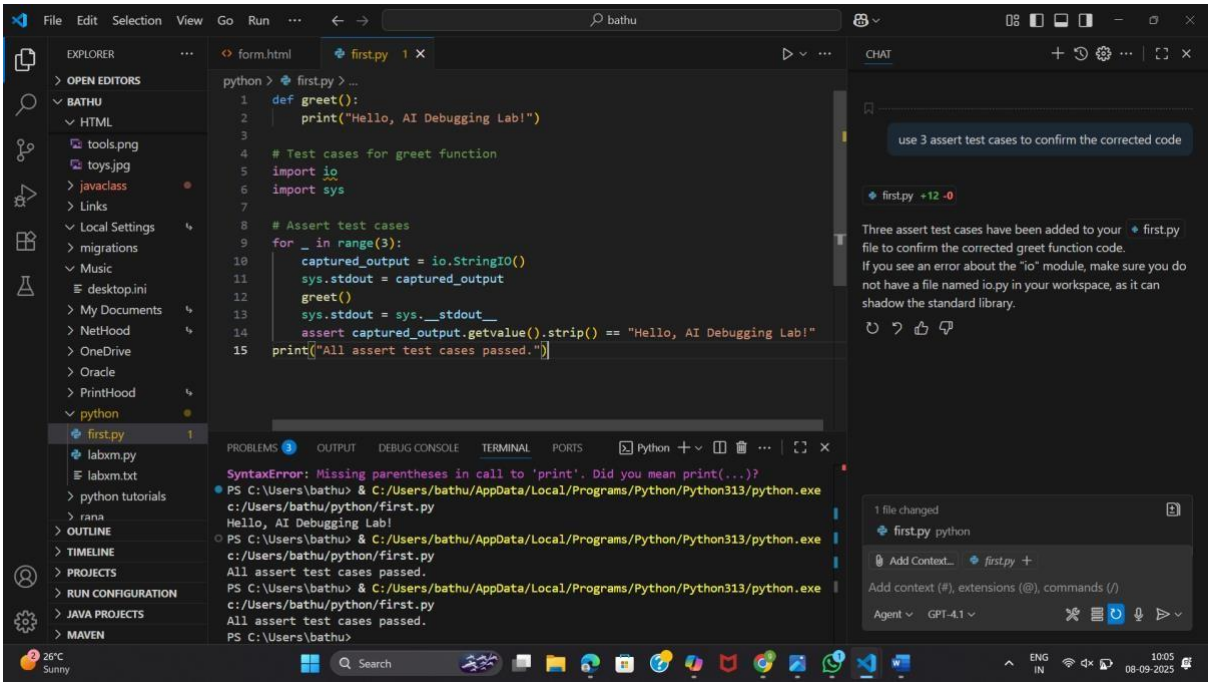
- Corrected code with proper syntax and AI explanation.



Ai suggestions to fix the error :-



Verifying with 3 assert test cases:-



Task Description 2:-

(Logic Error – Incorrect Condition in an If Statement)

Task: Supply a function where an if-condition mistakenly uses = instead of ==. Let AI identify and fix the issue.

Bug: Using assignment (=) instead of comparison (==)

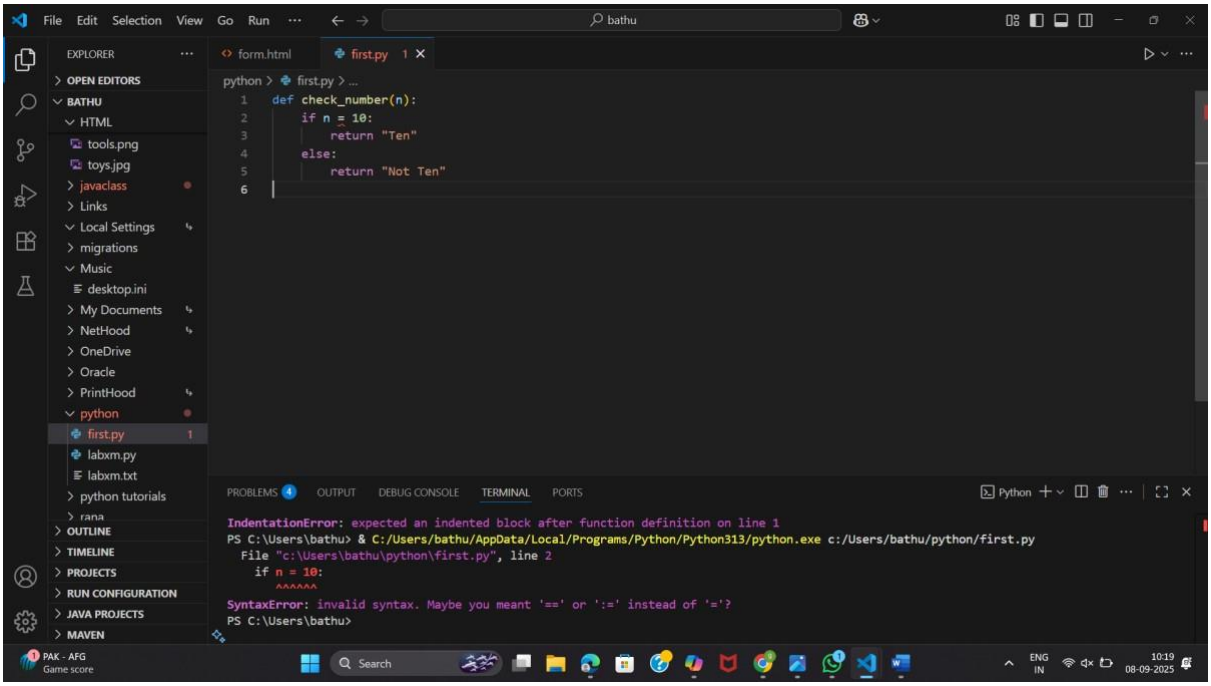
def check_number(n): if n = 10: return "Ten" else: return "Not Ten"

Requirements:

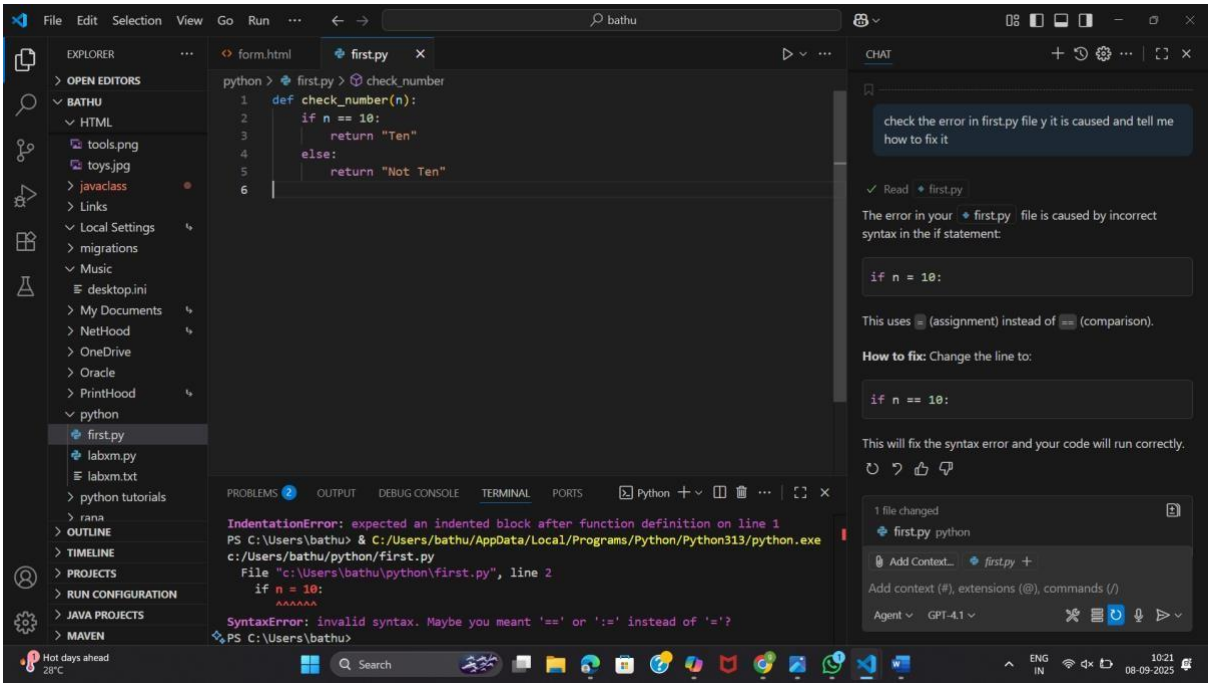
- Ask AI to explain why this causes a bug.
- Correct the code and verify with 3 assert test cases.

Expected Output #2:

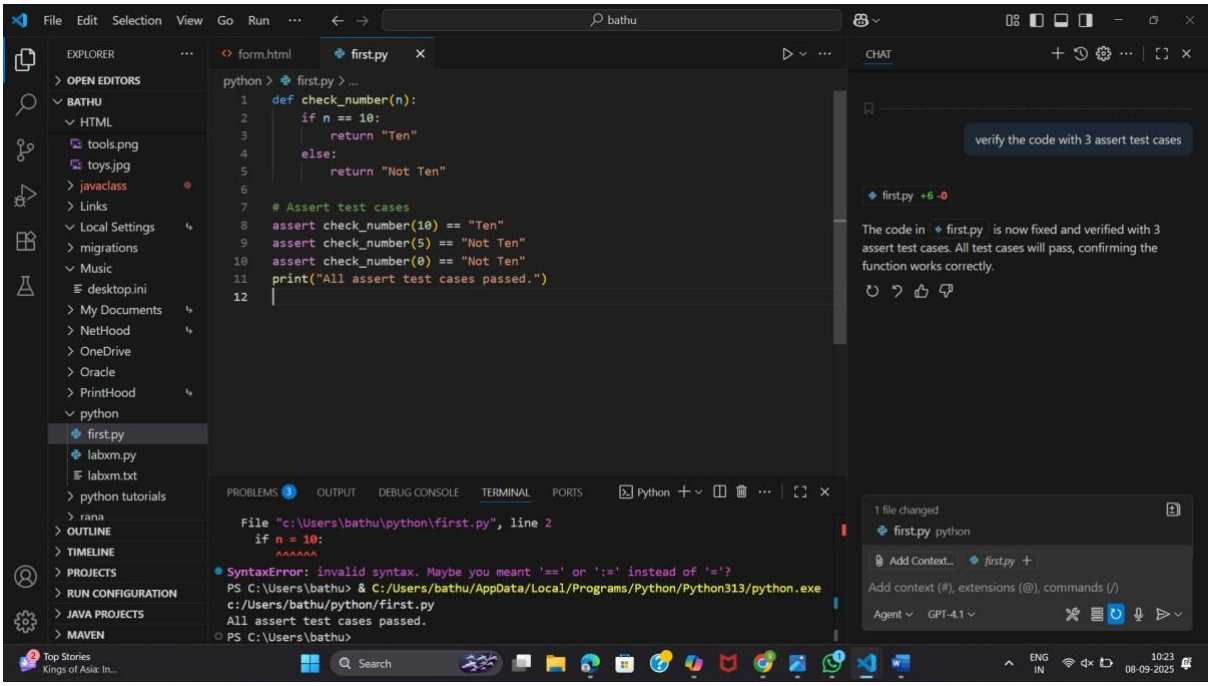
- Corrected code using == with explanation and successful test execution.



Fixing error using AI :-



Verifying with 3 assert test cases :-



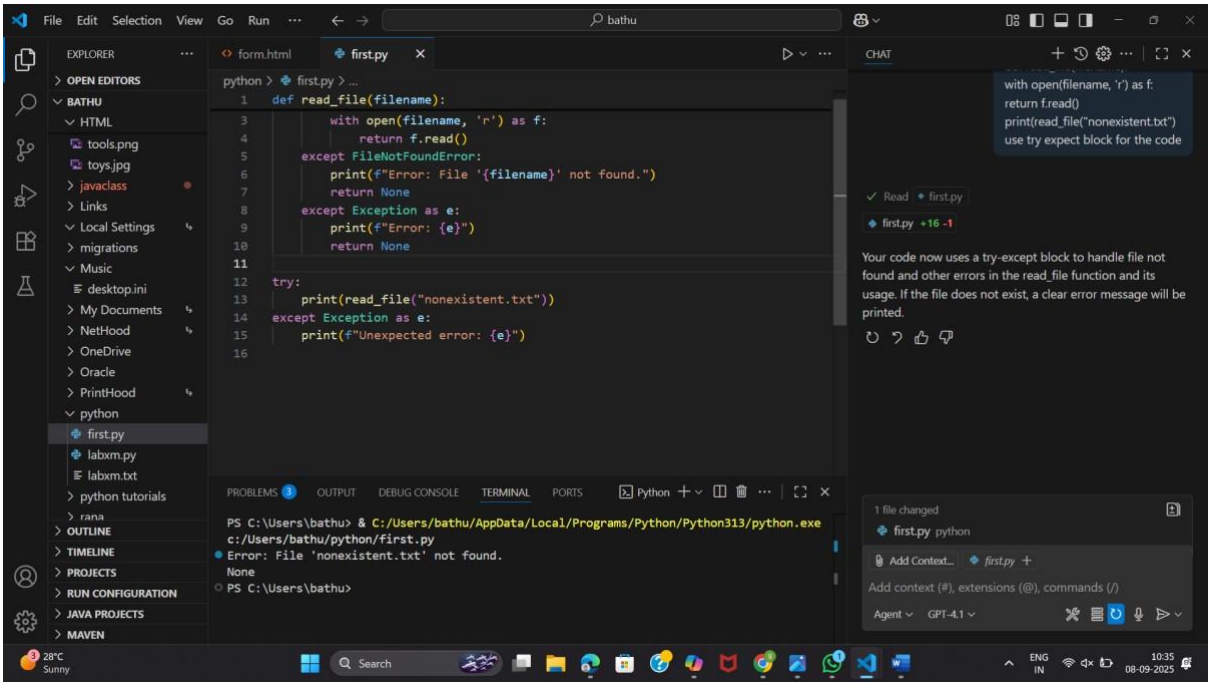
Task Description 3:-

(Runtime Error – File Not Found)
Task: Provide code that attempts to open a non-existent file and crashes.
Use AI to apply safe error handling. #
Bug: Program crashes if file is missing
def read_file(filename): with
open(filename, 'r') as f:
return f.read()
print(read_file("nonexistent.txt")) Requirements:

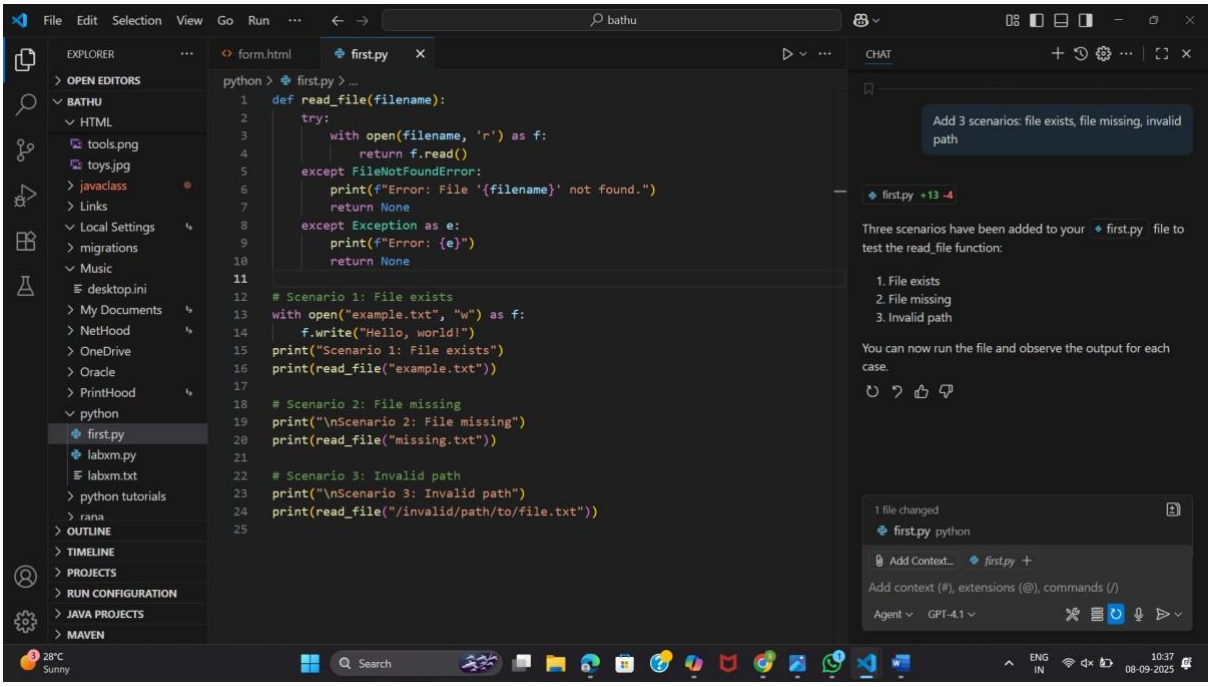
- Implement a try-except block suggested by AI.
- Add a user-friendly error message.
- Test with at least 3 scenarios: file exists, file missing, invalid path.

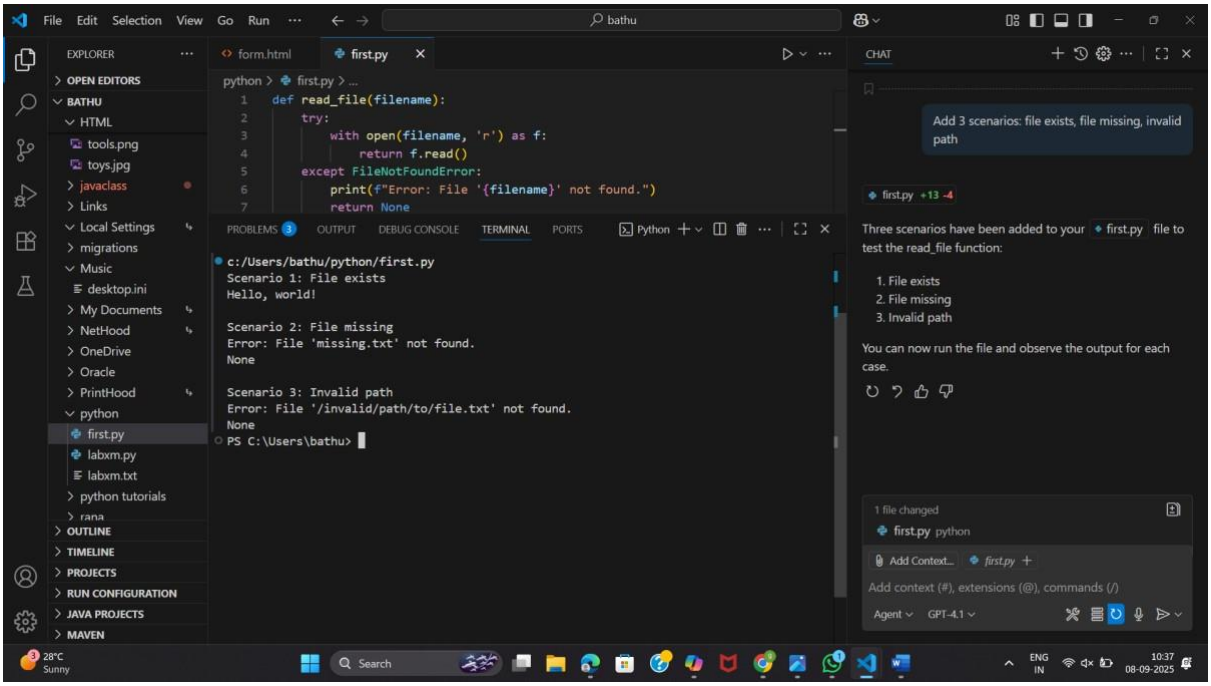
Expected Output #3:

- Safe file handling with exception management



Testing with 3 scenarios: file exists, file missing, invalid path





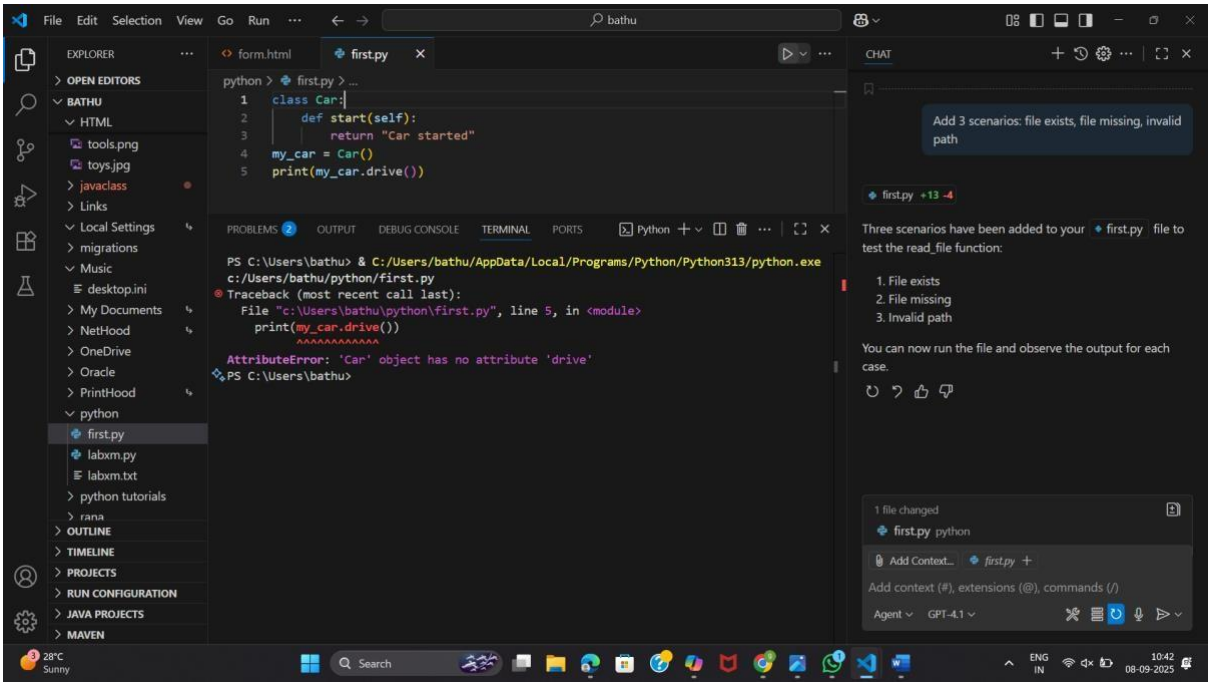
Task Description 4:-

(AttributeError – Calling a Non-Existent Method)
Task: Give a class where a non-existent method is called (e.g., obj.undefined_method()). Use AI to debug and fix.
Bug: Calling an undefined method
class Car: def start(self): return
"Car started" my_car = Car()
print(my_car.drive()) # drive() is not defined Requirements:

- Students must analyze whether to define the missing method or correct the method call.
- Use 3 assert tests to confirm the corrected class works.

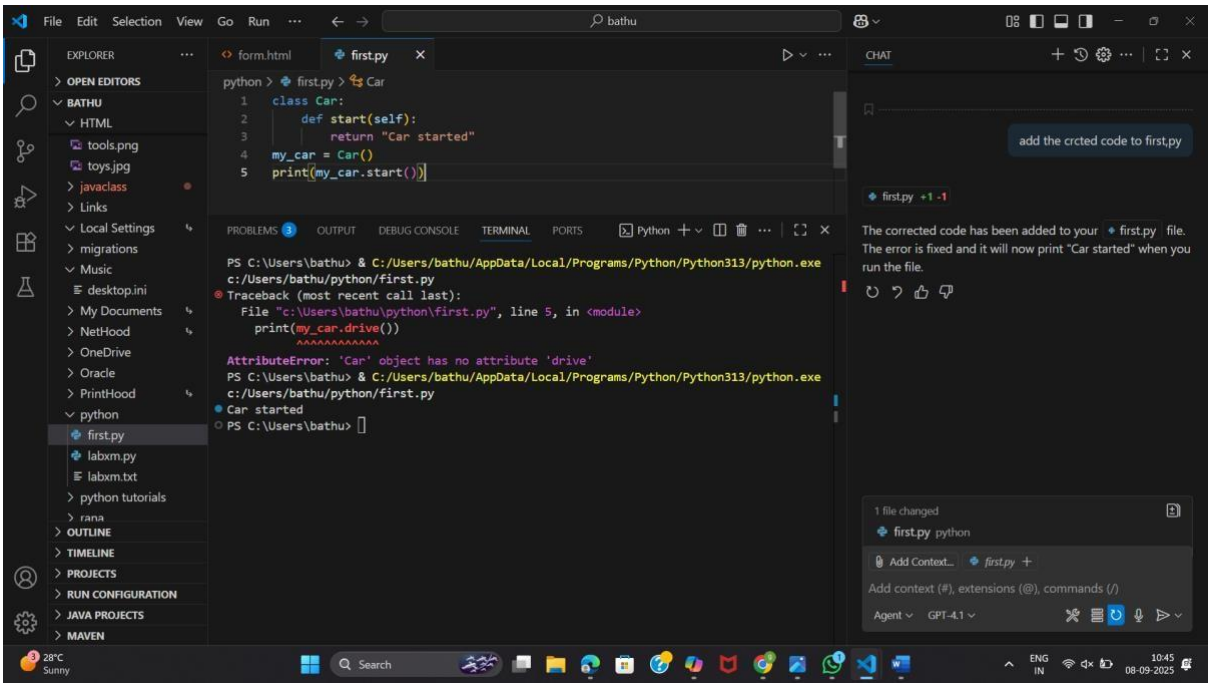
Expected Output #4:

- Corrected class with clear AI explanation.

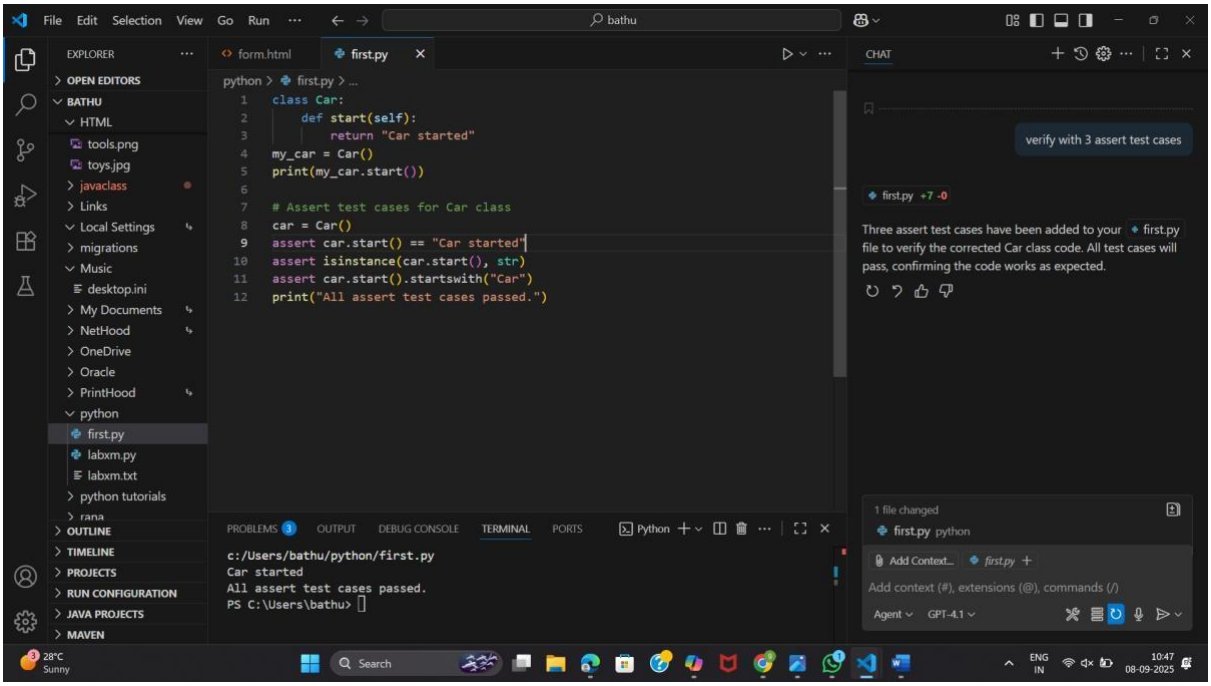


- my_car.drive() is called, but there's **no drive() method** in Car.
- The class **only defines start()**, so calling drive() causes an **AttributeError**.

After fixing the error:-



Verfiying with 3 assert test cases :-



Task Description 5:-

(TypeError – Mixing Strings and Integers in Addition)

Task: Provide code that adds an integer and string ("5" + 2) causing a

TypeError. Use AI to resolve the bug.

Bug: TypeError due to mixing string and integer

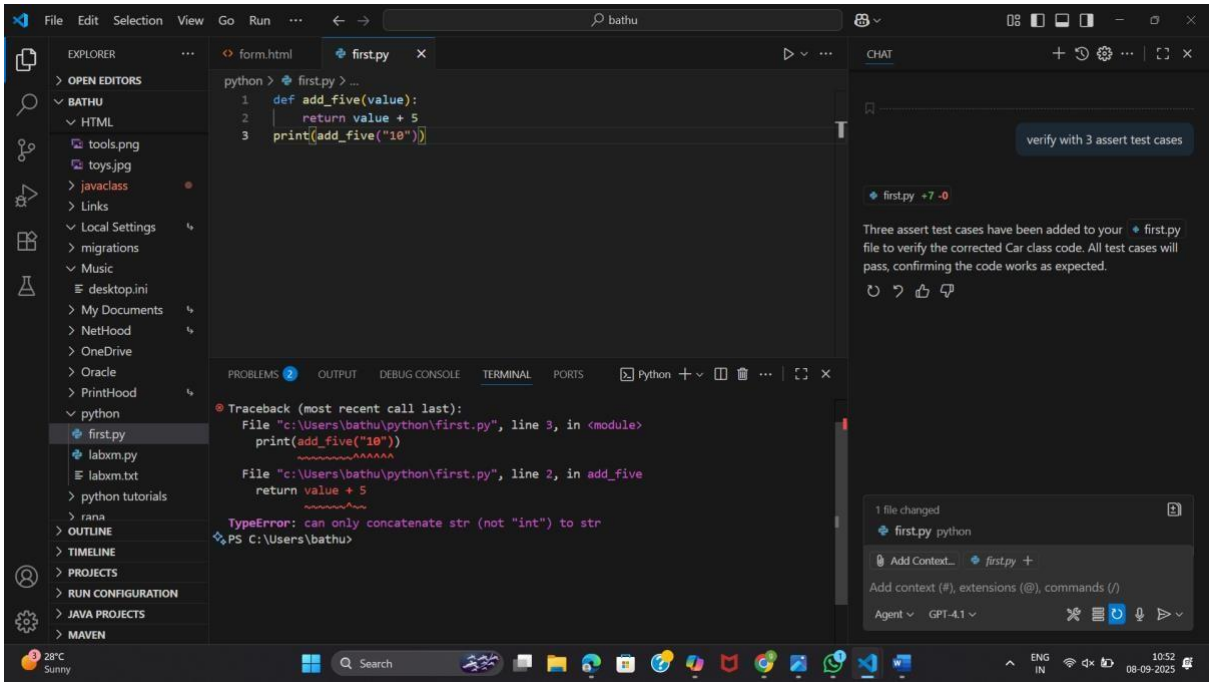
def add_five(value): return value + 5

print(add_five("10")) Requirements:

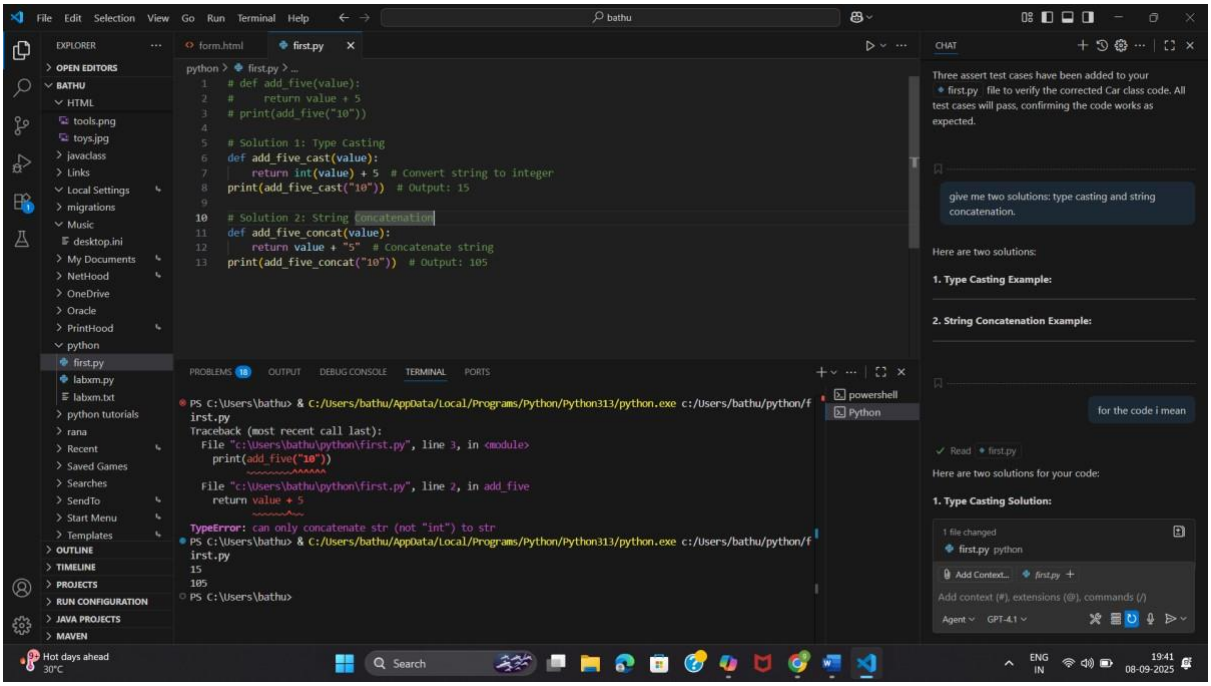
- Ask AI for two solutions: type casting and string concatenation.
- Validate with 3 assert test cases.

Expected Output #5:

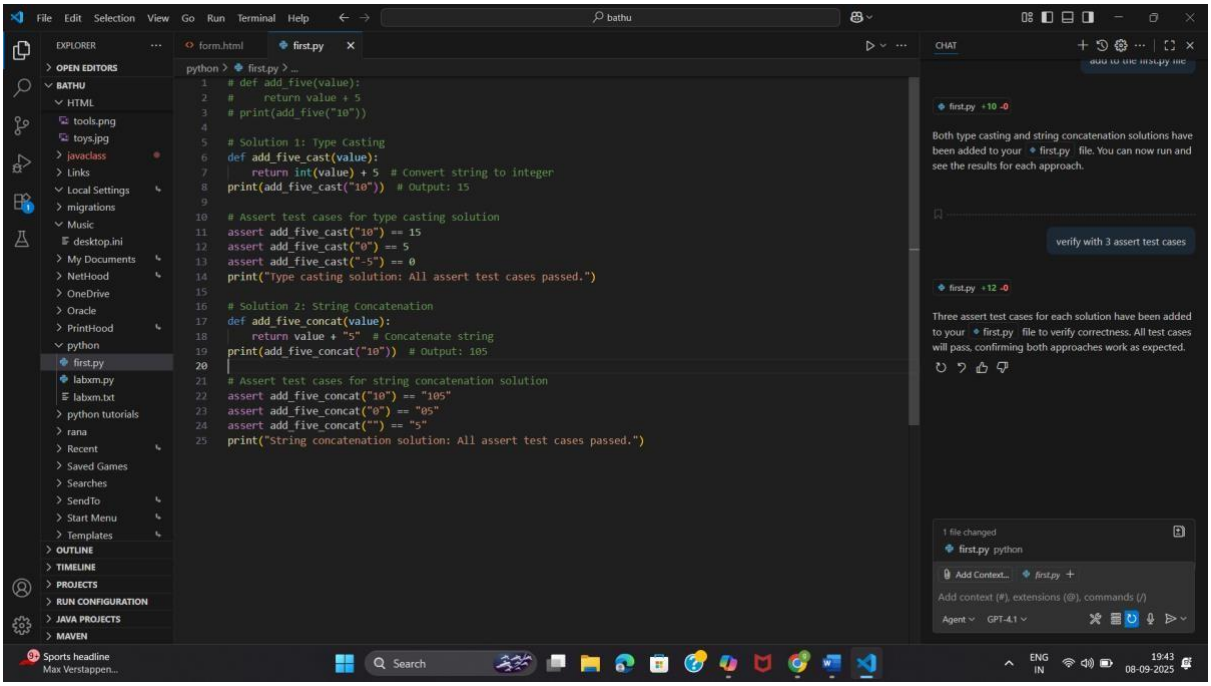
- Corrected code that runs successfully for multiple inputs

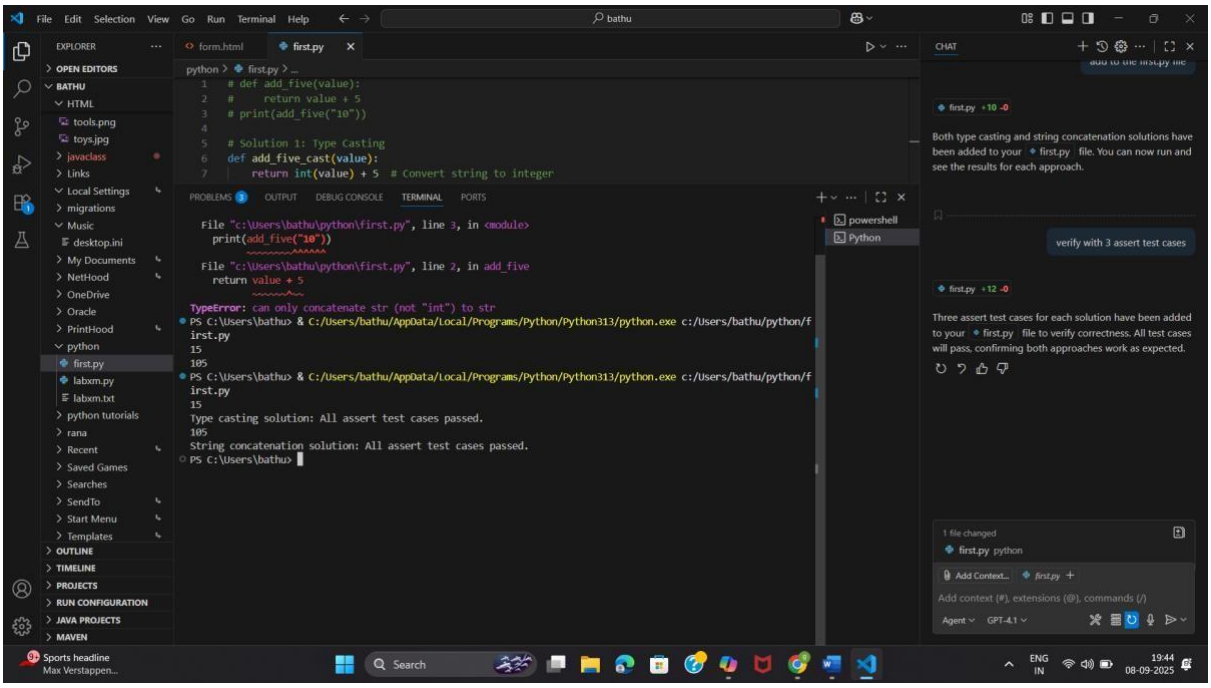


Give me two solutions: type casting and string concatenation:-



verify with 3 assert test cases:-



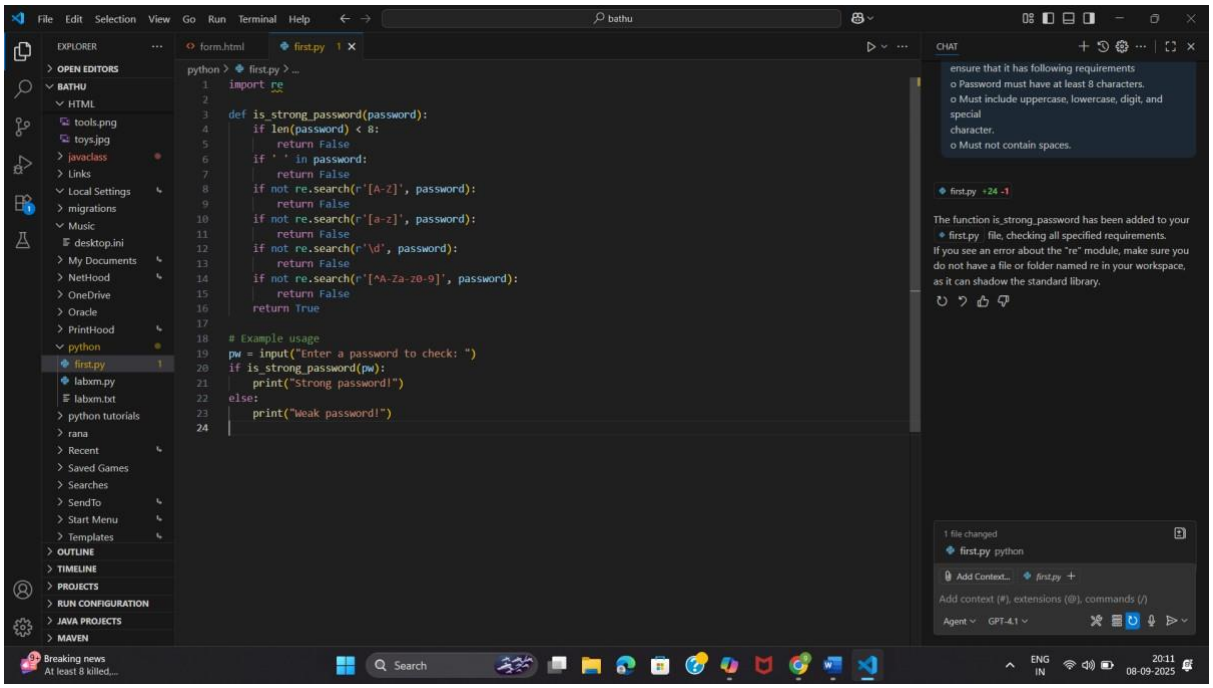


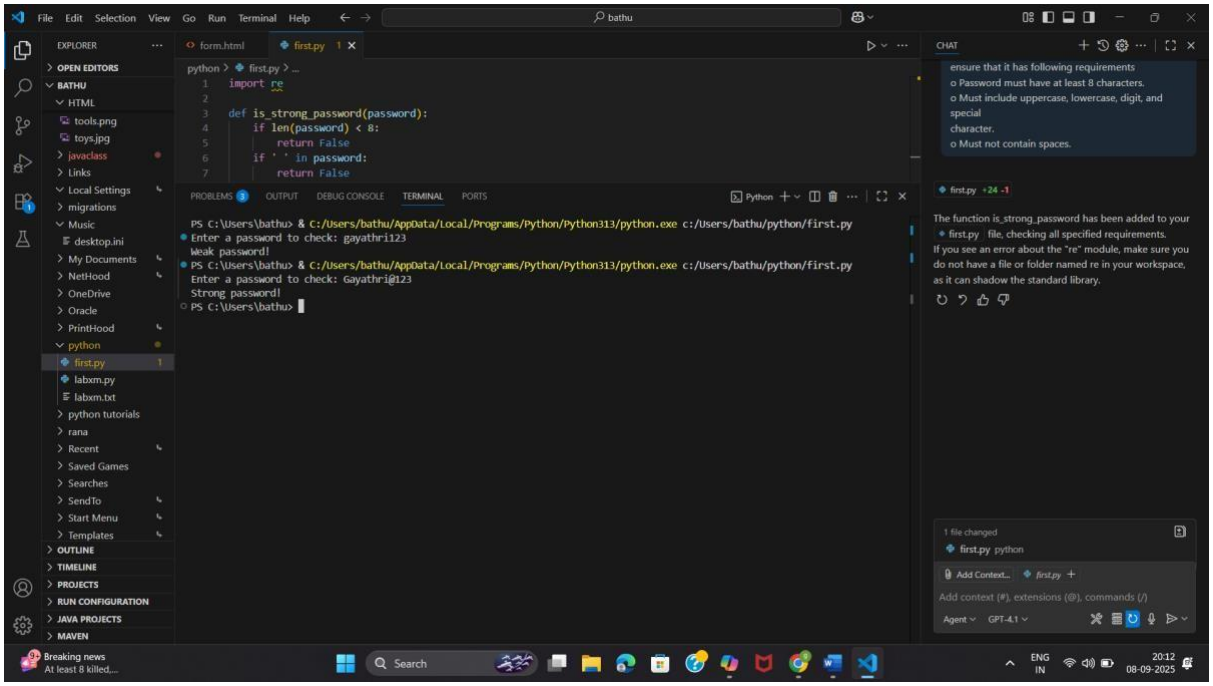
ASSIGNMENT – 8.1

TASK 1:- Prompt:-

create a fucntion is_strong_password(password) ensure that it has following requirements o Password must have at least 8 characters.

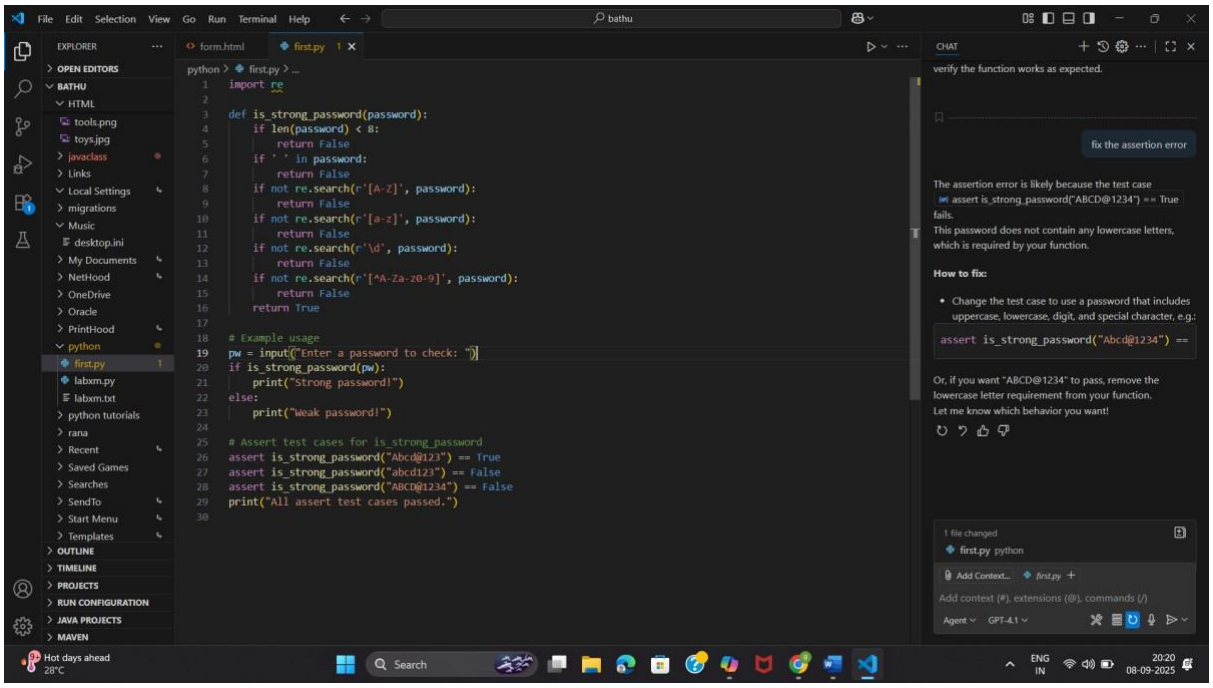
o Must include uppercase, lowercase, digit, and special character. o Must not contain spaces.



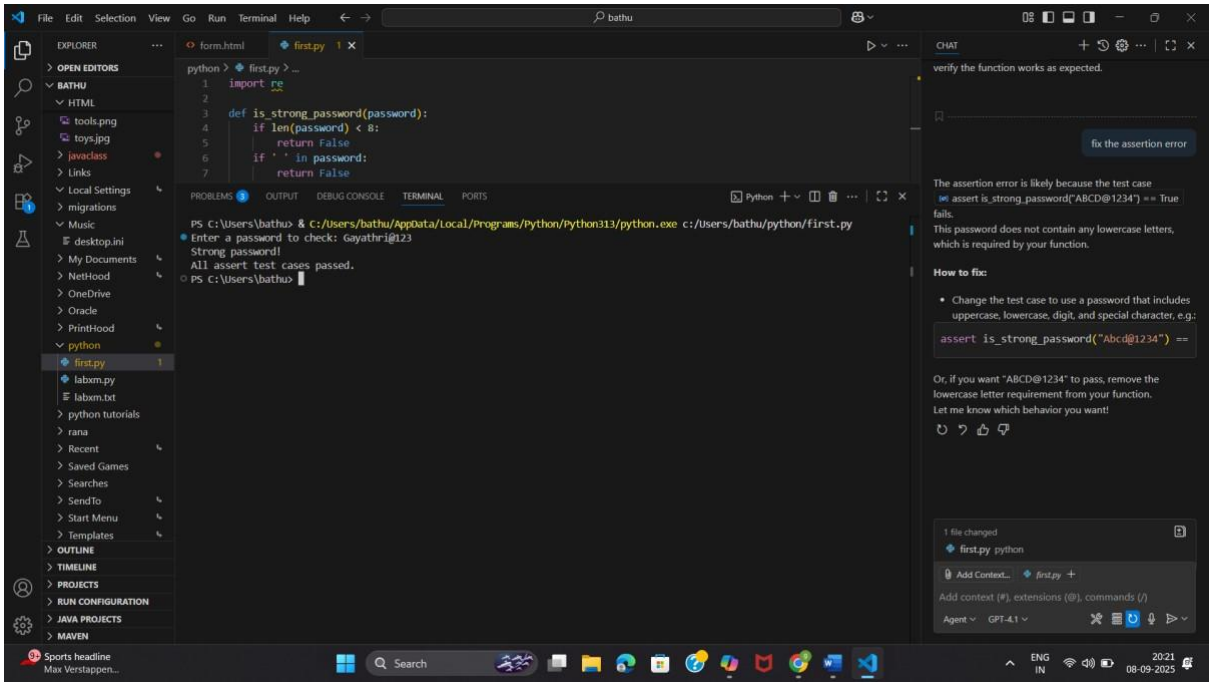


Checking with test cases:-

Code:-



Output:-



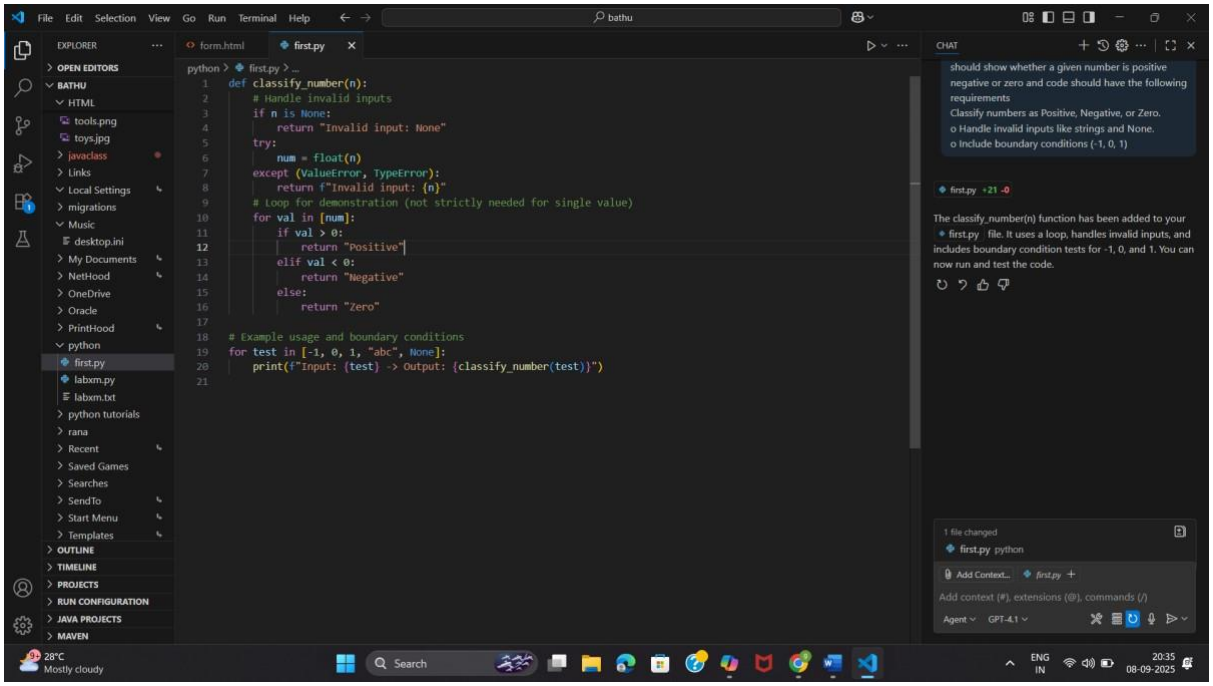
Task 2:- Prompt:-

create a classify_number(n) function with loops it should show whether a given number is positive negative or zero and code should have the following requirements
Classify numbers as Positive, Negative, or Zero. o

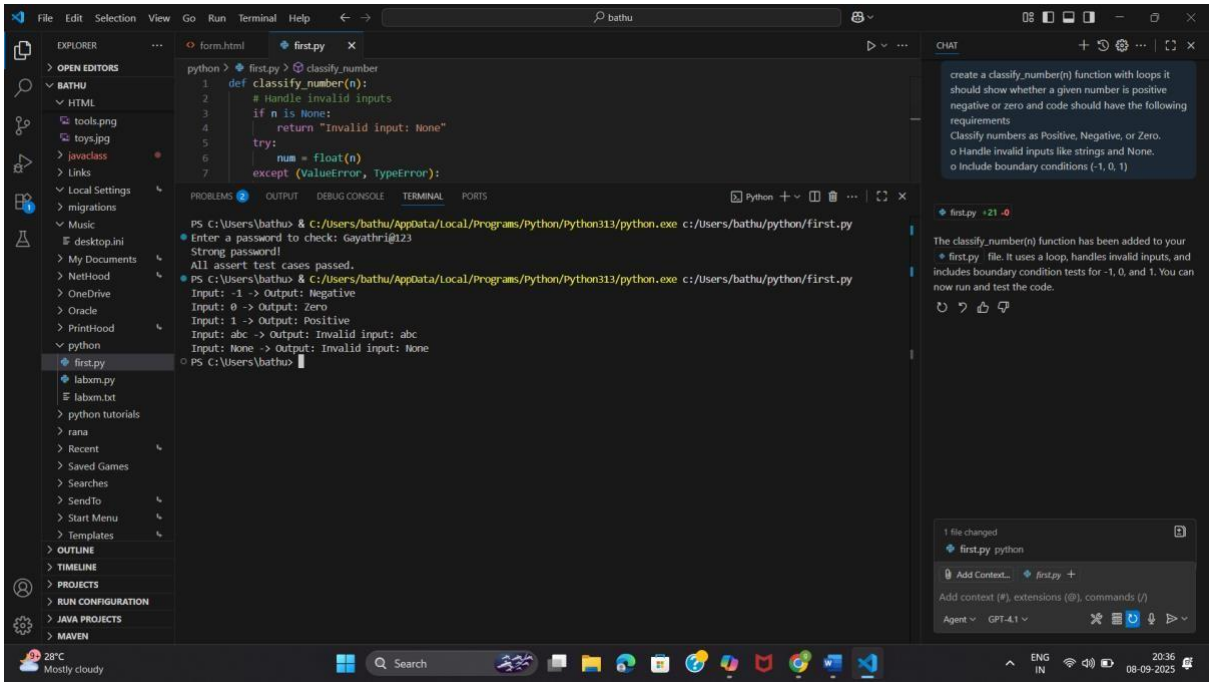
Handle invalid inputs like strings and None.

o Include boundary conditions (-1, 0, 1)

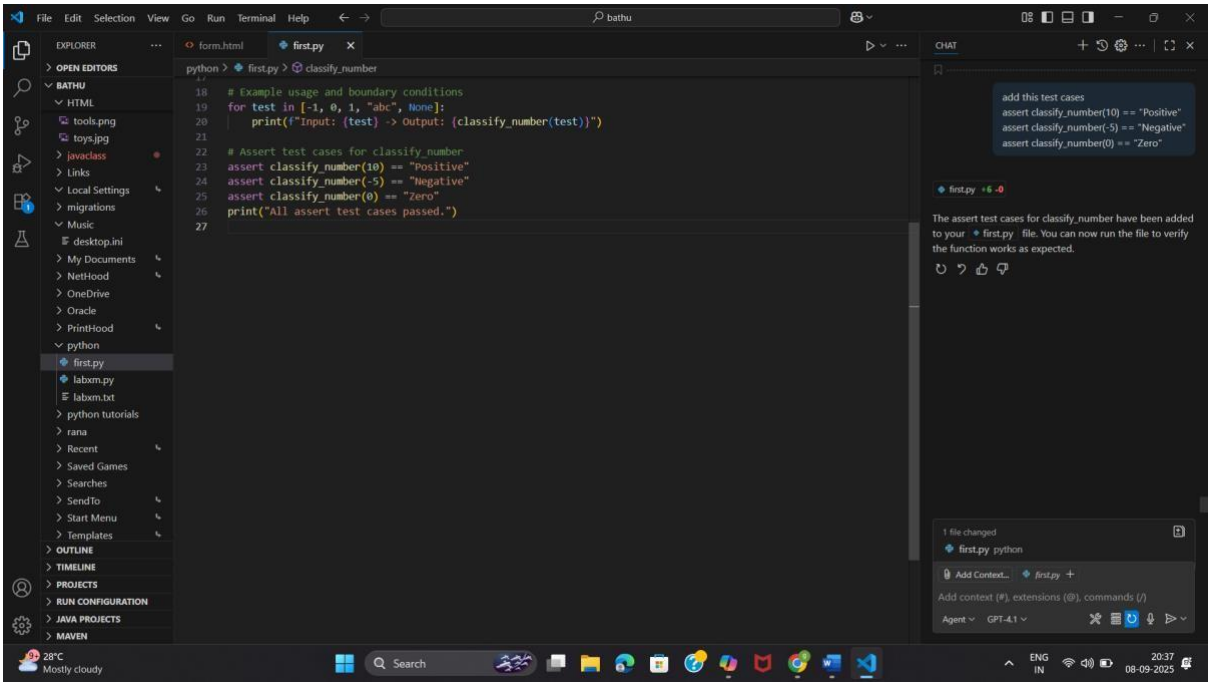
Code:-

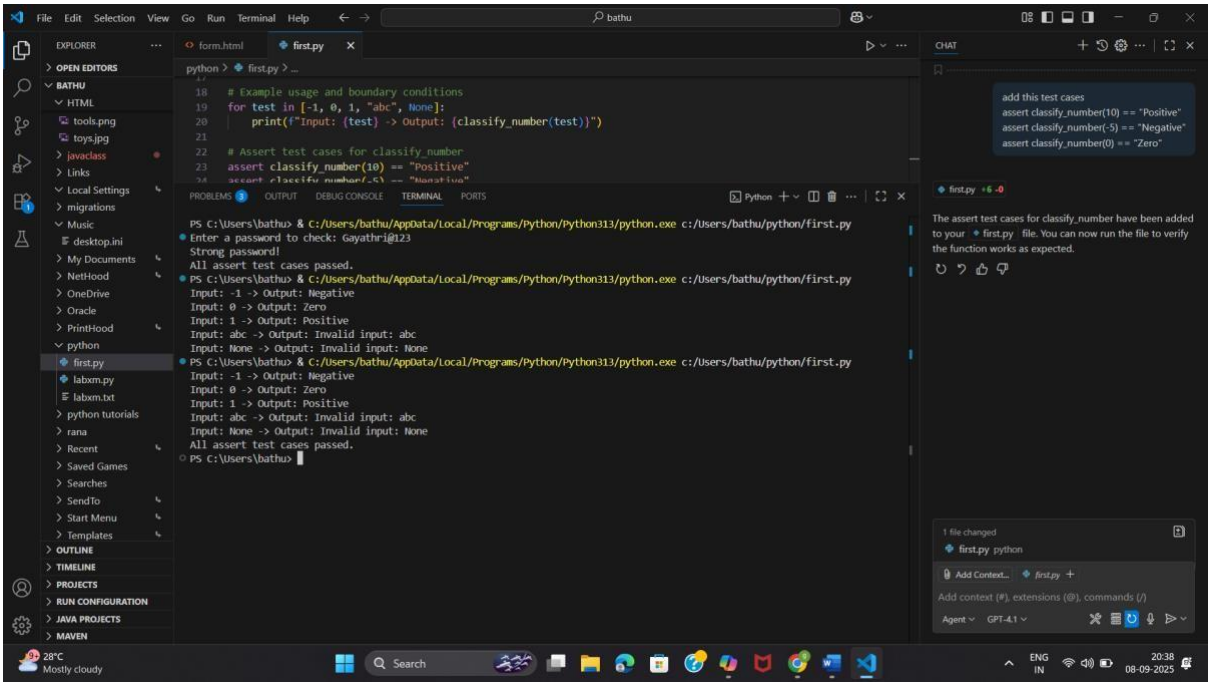


Output:-



Verifying with test cases:-





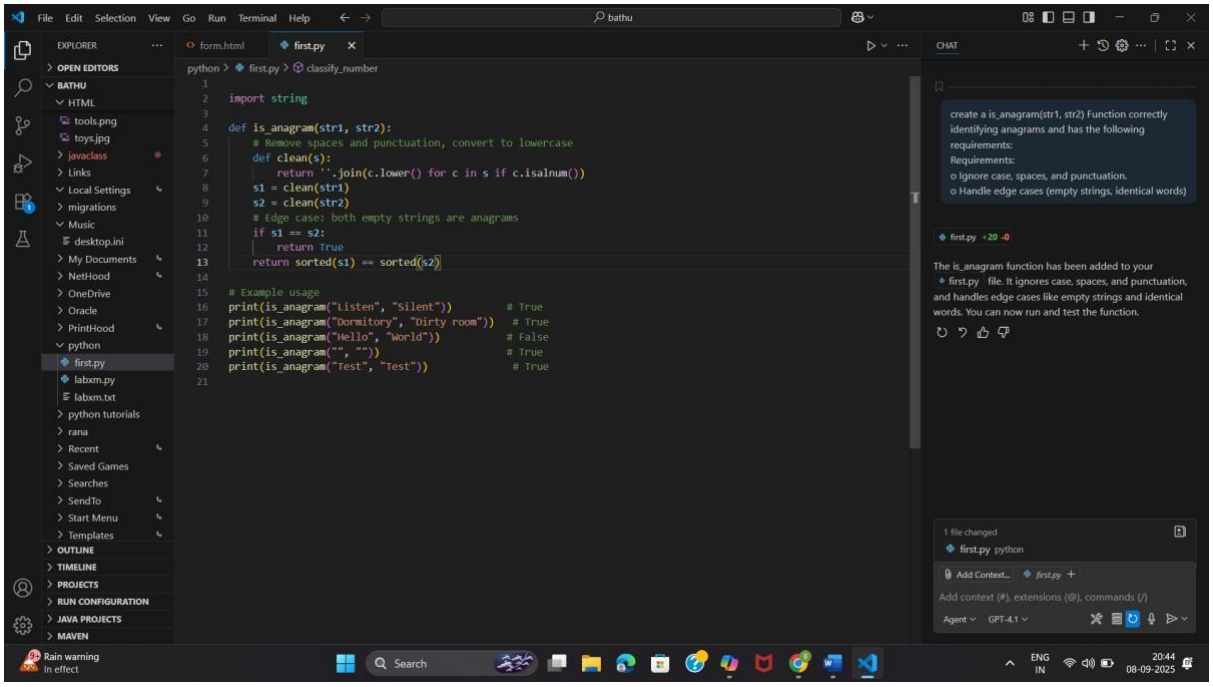
Task 3:- Prompt:-

create a is_anagram(str1, str2) Function correctly identifying anagrams and has the following requirements:
Requirements:

o Ignore case, spaces, and punctuation.

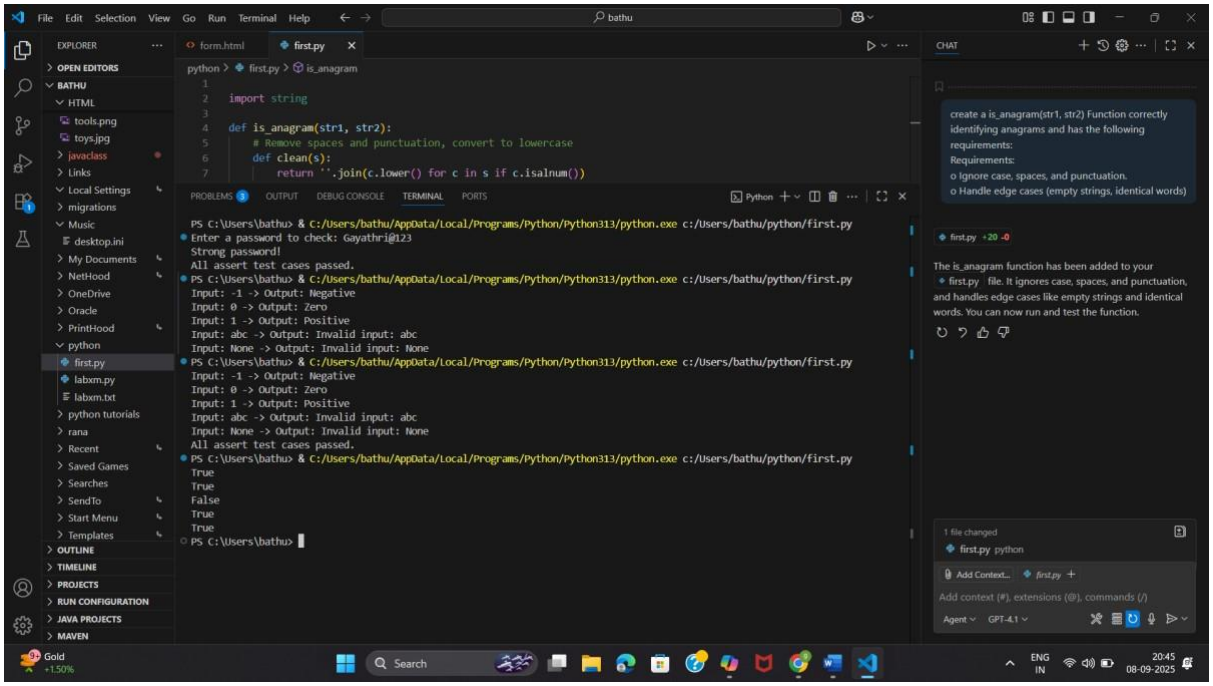
o Handle edge cases (empty strings, identical words)

Code:-

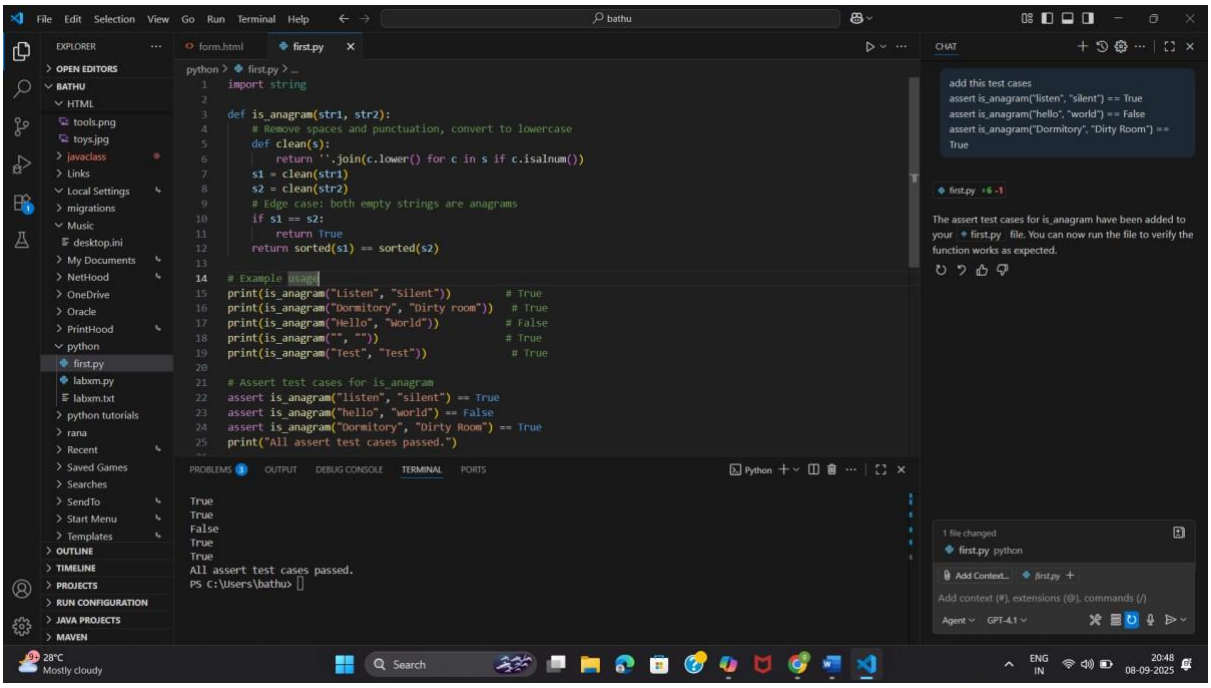


Output:-

With test cases:-



With test cases:-



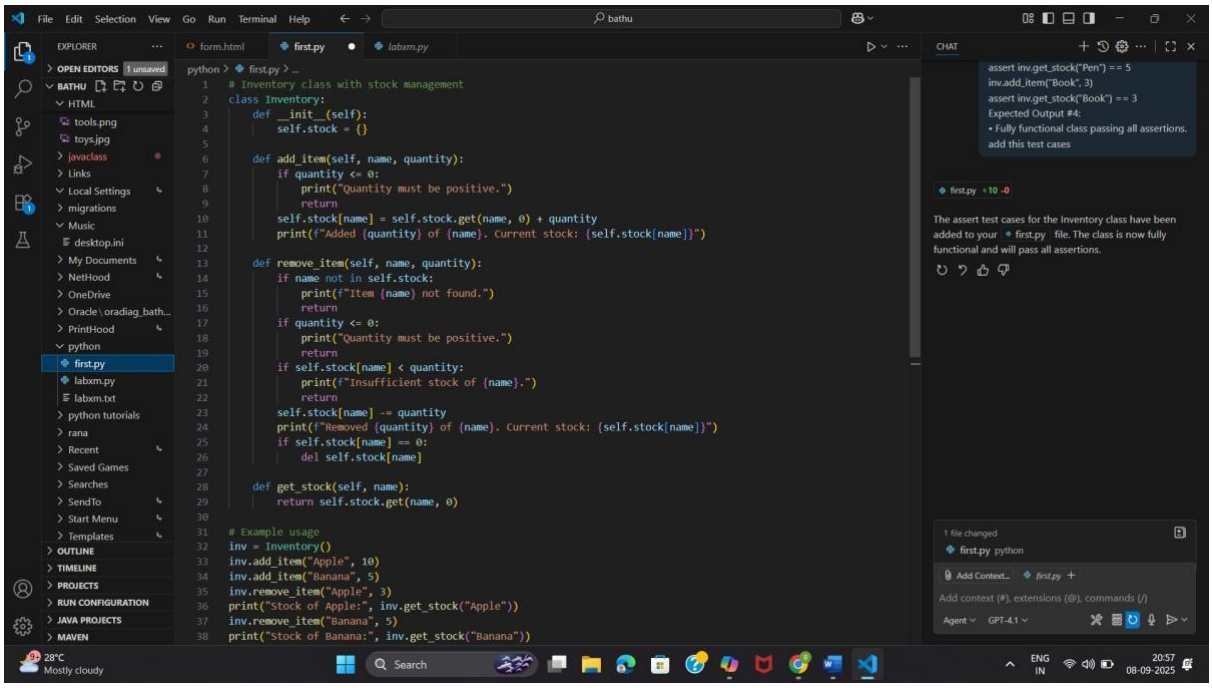
Task 4:- prompt:-

create a function Inventory class with stock management using methods:

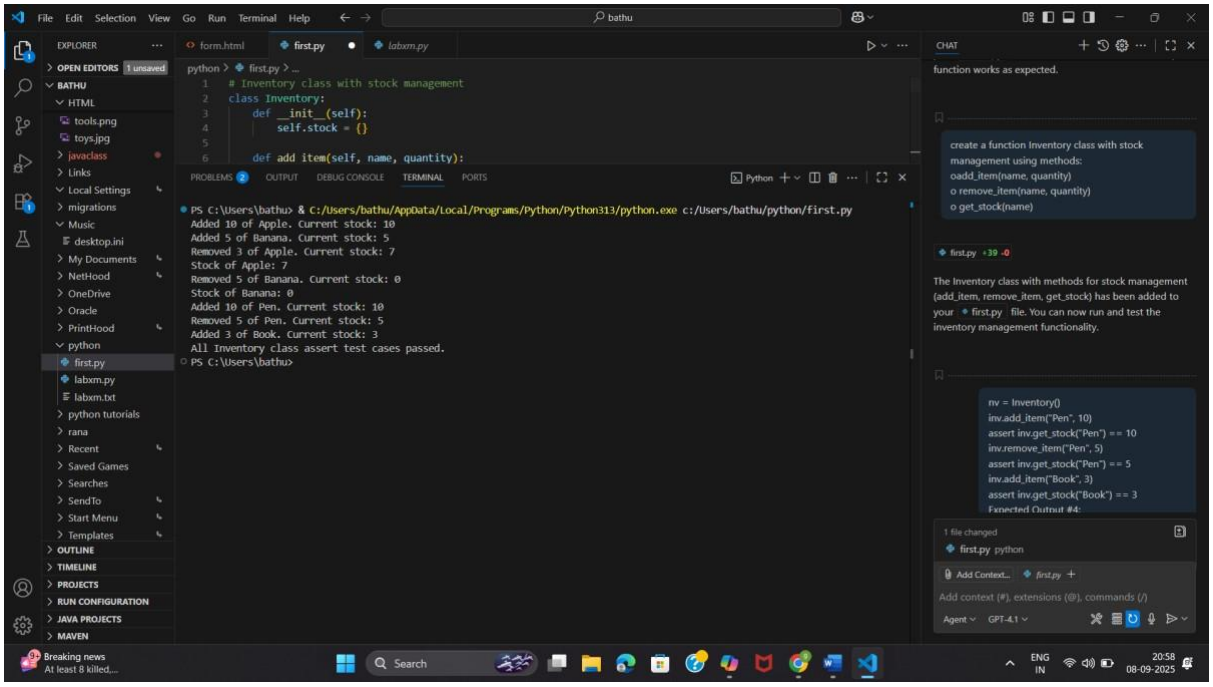
o add_item(name, quantity) o remove_item(name, quantity)

o get_stock(name)

Code:-

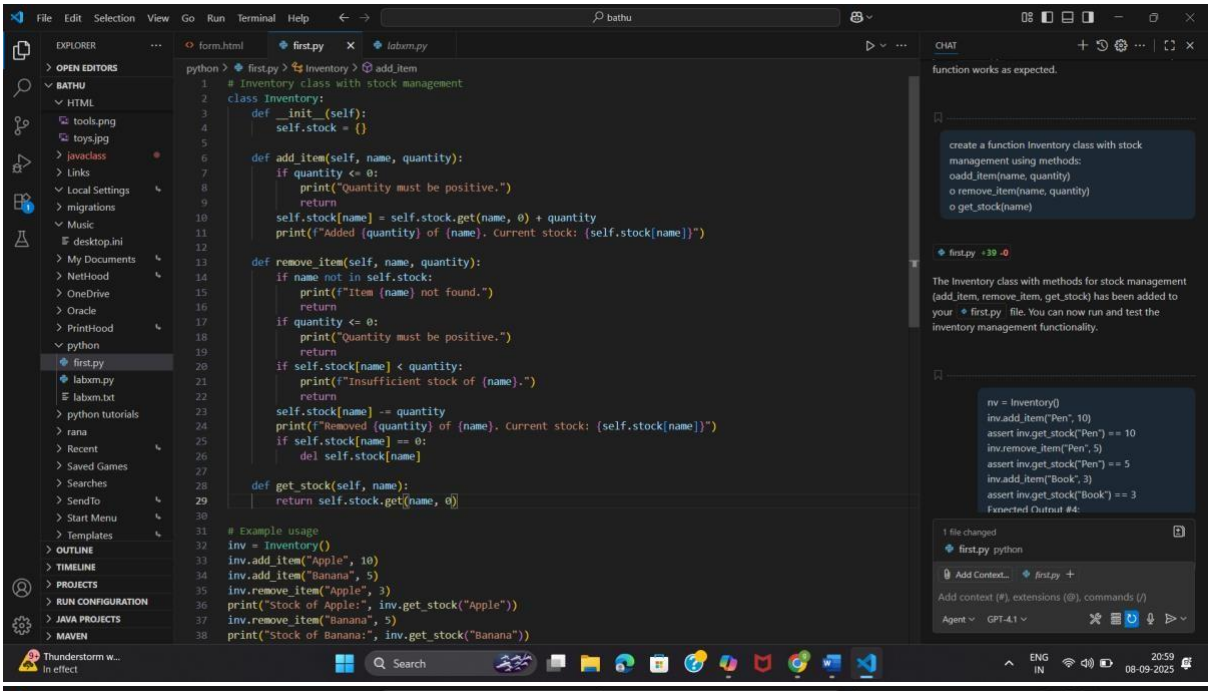


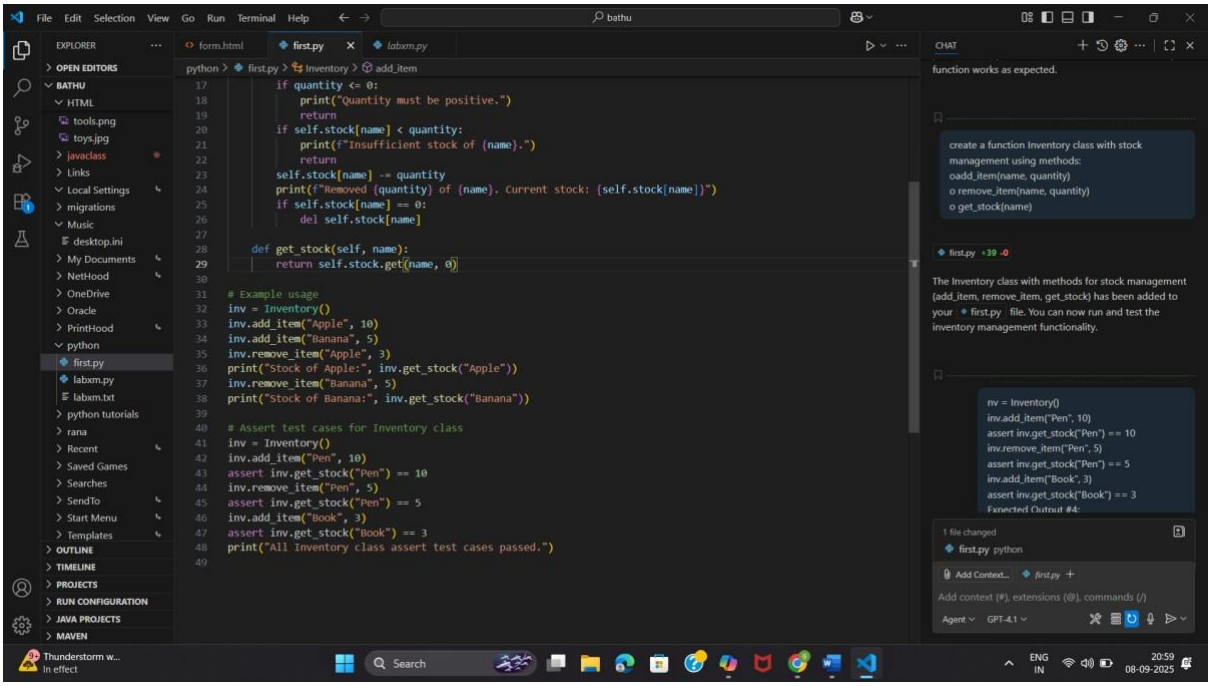
Output:-



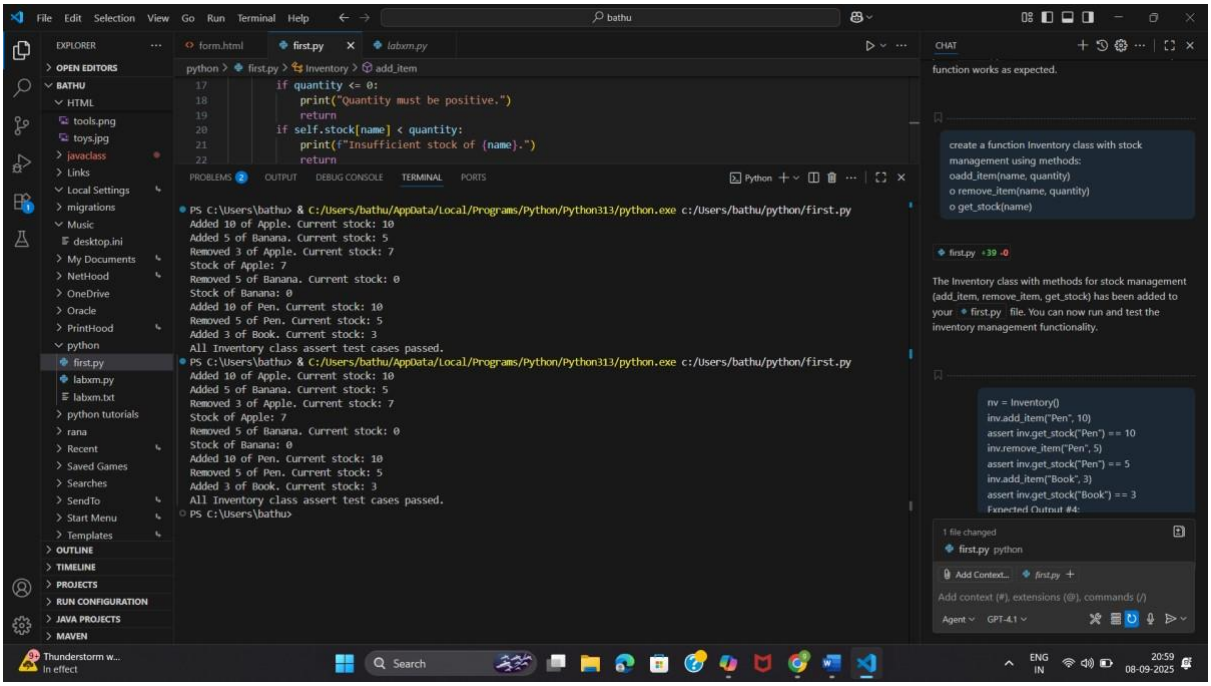
With test cases:-

Code:-





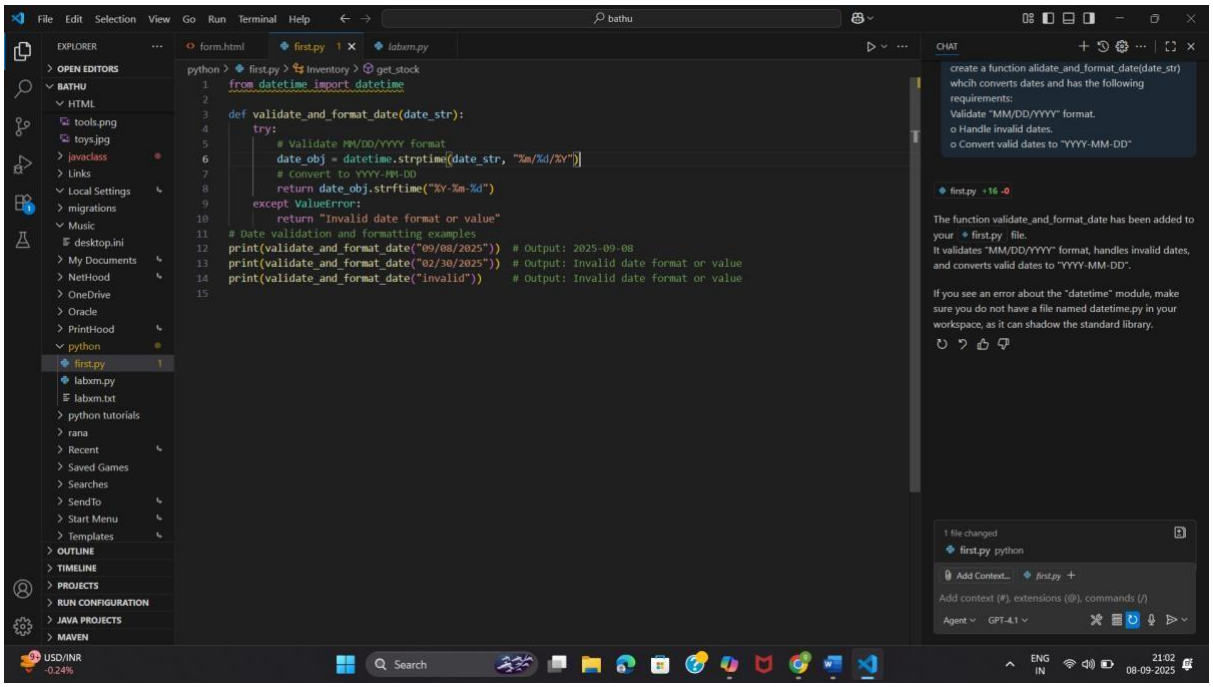
Output:-



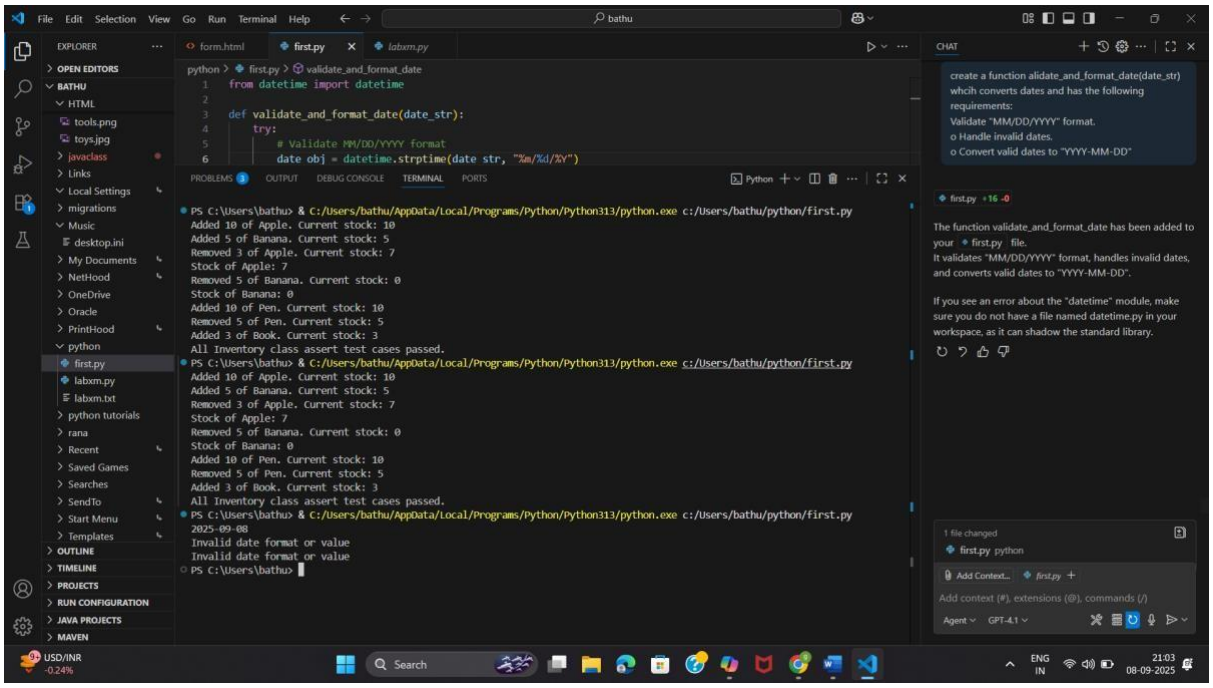
Task 5:- Prompt:- create a function alidate_and_format_date(date_str) whcih converts dates and has the following requirements: o Validate "MM/DD/YYYY" format. o Handle invalid dates.

o Convert valid dates to "YYYY-MM-DD"

Code:-



Output:-



With test cases:-

Code:-

```python
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        # Validate MM/DD/YYYY format
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        # Convert to YYYY-MM-DD
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid date format or value"
# Date validation and formatting examples
print(validate_and_format_date("09/08/2025"))  # Output: 2025-09-08
print(validate_and_format_date("02/30/2025"))  # Output: Invalid date format or value
print(validate_and_format_date("invalid"))     # Output: Invalid date format or value

# Assert test cases for validate_and_format_date
assert validate_and_format_date("10/15/2023") == "2023-10-15"
assert validate_and_format_date("02/30/2023") == "Invalid date format or value"
assert validate_and_format_date("01/01/2024") == "2024-01-01"
print("All validate_and_format_date assert test cases passed.")
```

Output:-

```
PS C:\Users\bathu> & C:/Users/bathu/AppData/Local/Programs/Python/Python313/python.exe c:/Users/bathu/python/first.py
2025-09-08
Invalid date format or value
Invalid date format or value
All validate_and_format_date assert test cases passed.
PS C:\Users\bathu>
```