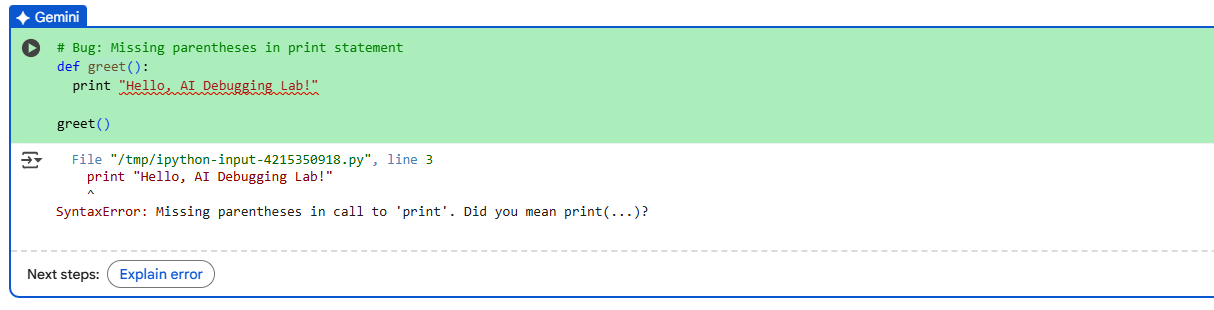
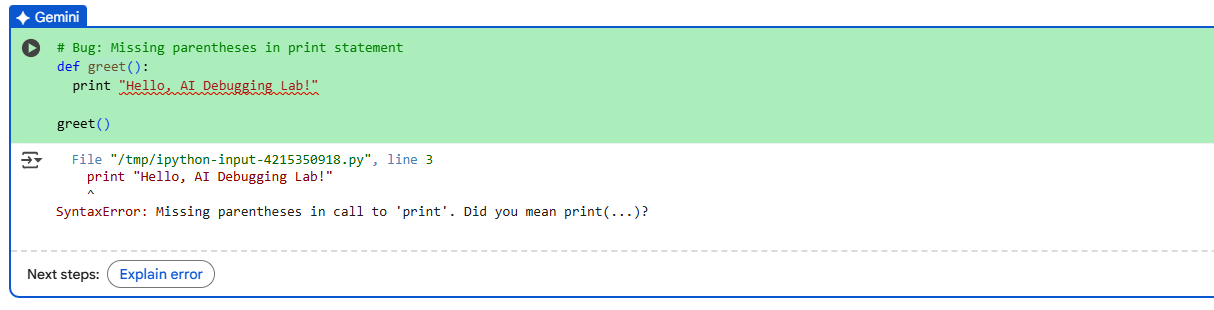
LAB ASSIGNMENT 7.1

Task Description #1 (Syntax Errors – Missing Parentheses in Print Statement)

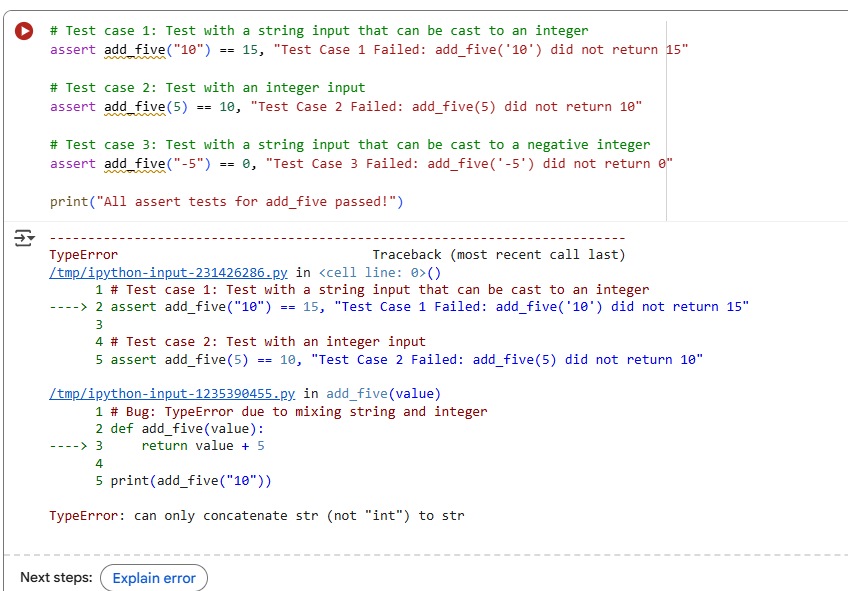
Code:



Output:



3 assert test cases



Observation:

**1.**In Python , must put brackets around what you want to print: print("...").

2.If you run the code, Python will show a red error saying something's wrong with the print line.

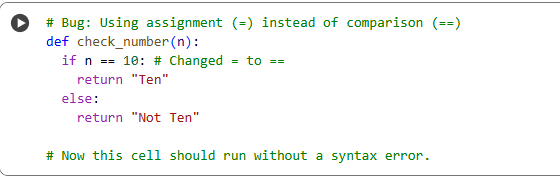
**3.**Just add brackets: change print "Hello" to print("Hello").

**4.**Python changed how print works, so using the right format helps your code run smoothly.

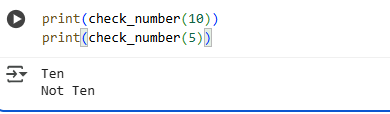
5.The code uses print "..." which worked in older Python versions but not in newer ones.

TASK DESCRIPTION#2(Logic Error – Incorrect Condition in an If Statement)

Code:



Output:



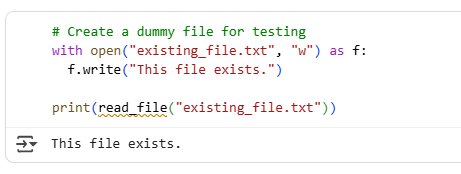
Observation:

1. The code uses = instead of == in the if statement.
2. Means assignment= is for giving a value to something, not for checking if two things are equal.
3. Python gets confused ,Python expects a condition in if, but sees an assignment—so it throws an error.
4. To compare values, you should use ==, like if n == 10.
5. Changing = to == lets Python understand the condition and run the code properly.

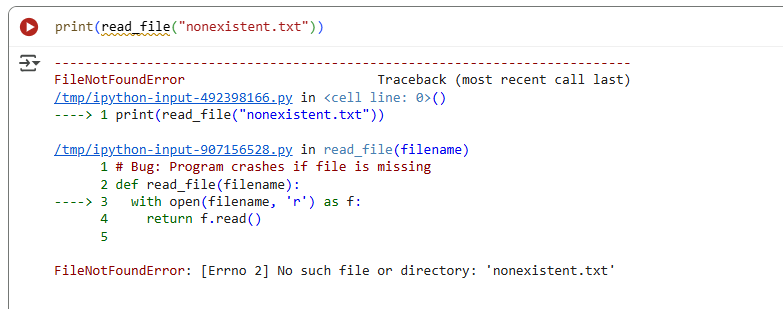
TASK DESCRIPTION#3 (Runtime Error – File Not Found)

Code:

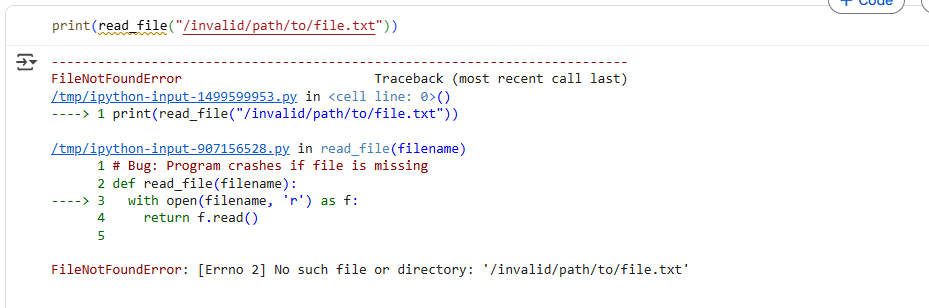
Scenario 1: File exists



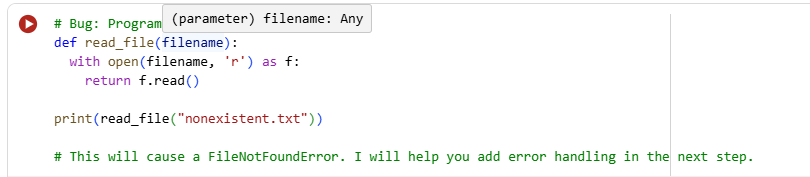
Scenario 2: File missing (already demonstrated, but let's run it again)



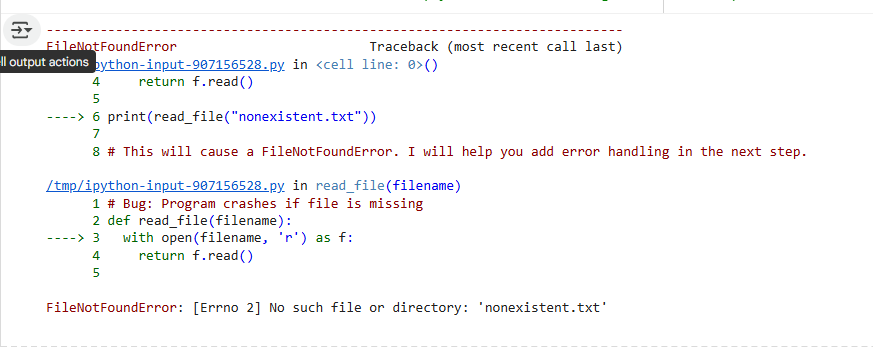
Scenario 3: Invalid path



Code:



Code output:

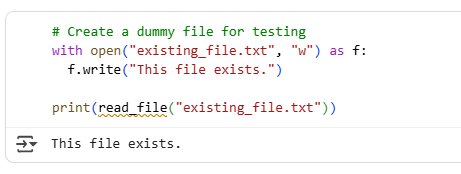


Observation:

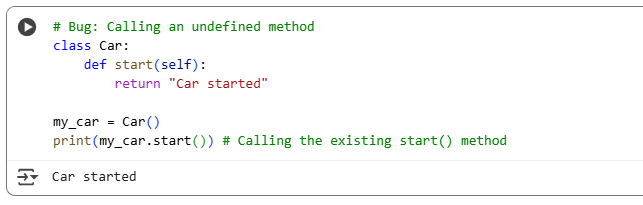
1. **No More Crashing:** The main thing is that the code now doesn't break when it can't find a file. Before, it would just stop working with an error. Now, it tells you nicely that the file wasn't found.
2. **Finding Files Works:** When the file is actually there, the code can still open it and read what's inside, just like it should.
3. We first saw the code fail on purpose to understand why we needed to fix it.
4. We tested the code in three clear situations (file is there, file is missing, bad file name) so it's easy to see that the fix works in all those cases.
5. We also saw how the AI fixed a different kind of mistake earlier, where the code used = instead of == to check if two things were the same.

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()).

Code:

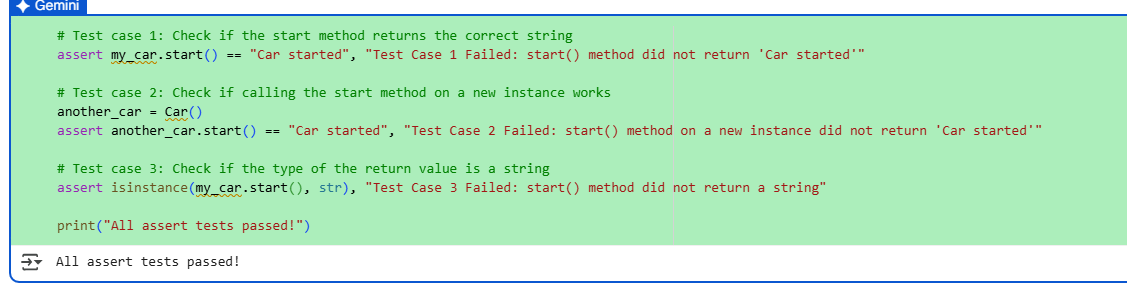


Output:



Using 3 assert tests to confirm the corrected class works

Code:



Observation:

1. **The Car:** We created a basic blueprint for a "Car". This blueprint has one action it knows how to do: start(). When you tell a car made from this blueprint to start(), it says "Car started".
2. We then made an actual car based on this blueprint. We called it my\_car.
3. The assert tests are like checking if the car does what it's supposed to do:

**a.test1:** We checked if *our* car (my\_car) says exactly "Car started" when we tell it to start(). If it doesn't, the test fails.

**b.Test 2:** We made *another* car (another\_car) and checked if *that* car also says "Car started" when we tell it to start(). This makes sure the blueprint works for any new car we make.

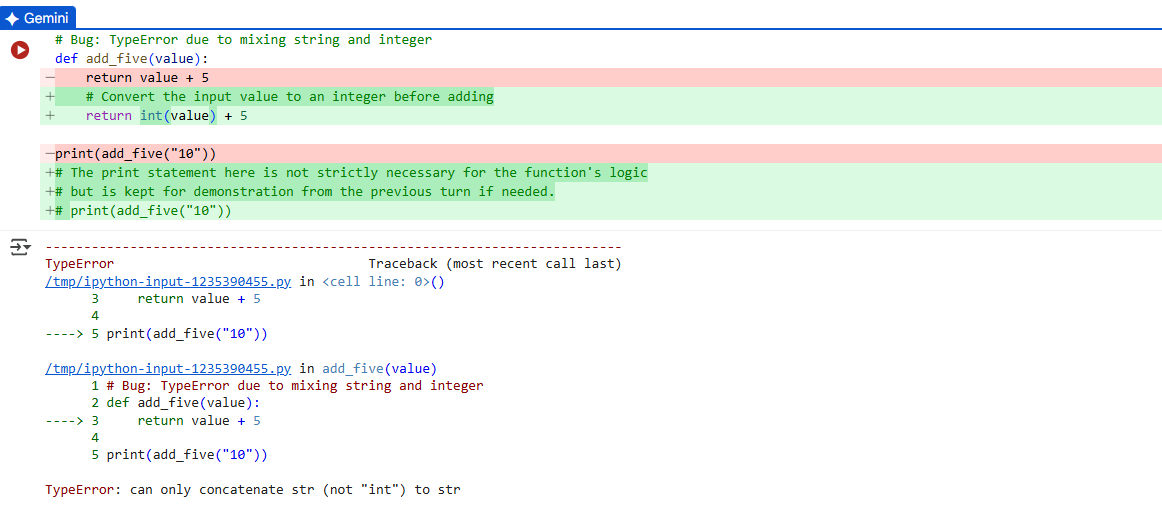
**c.Test 3:** We checked if what the car says when it starts is text (a "string"). This confirms the output is in the right format.

1. If you see "All assert tests passed!", it means all our checks confirmed that the car starts correctly and gives the right kind of message.

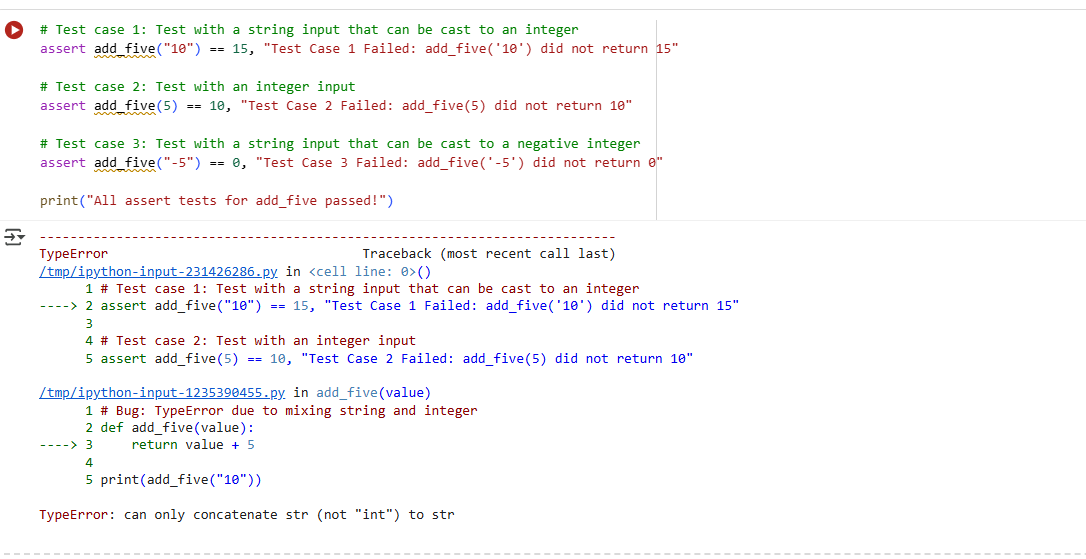
5. built a simple car with a start action, and the tests confirmed that the start action works correctly every time.

TASK DESCRIPTION #5 (TypeError – Mixing Strings and Integers in Addition)

Code:



Validate with 3 assert test cases



Observation:

1. **The Original Problem:** We had a function called add\_five that was supposed to add 5 to a number. But when we gave it a number as text (like "10"), it got confused because you can't directly add text and numbers in that way. This caused a TypeError.
2. **The Fix:** We told the function to first turn any text input into a whole number (int(value)) *before* adding 5 to it.
3. **The Tests:** The assert tests are like checking if our fixed function works for different kinds of inputs:

**a.Test 1 (Text "10"):** We checked if add\_five("10") gives us 15. It now does because the function first turns "10" into the number 10, and then 10 + 5 is 15.

**b.Test 2 (Number 5):** We checked if add\_five(5) gives us 10. It still works because adding 5 to the number 5 is straightforward.

**c.Test 3 (Negative Text "-5"):** We checked if add\_five("-5") gives us 0. It works because the function turns "-5" into the number -5, and then -5 + 5 is 0.

1. **All Tests Passed:** Seeing "All assert tests passed!" means our fix worked! The function can now correctly add 5 whether you give it a number as text or as an actual number.
2. the function how to understand numbers given as text before doing the math.