## NAME:HARSHA

## ROLL NO:2403A51352

## BATCH:14

## DATE:15-09-2025

ASSIGNMENT-7.1

**Task Description #1**

(Syntax Errors – Missing Parentheses in Print Statement)

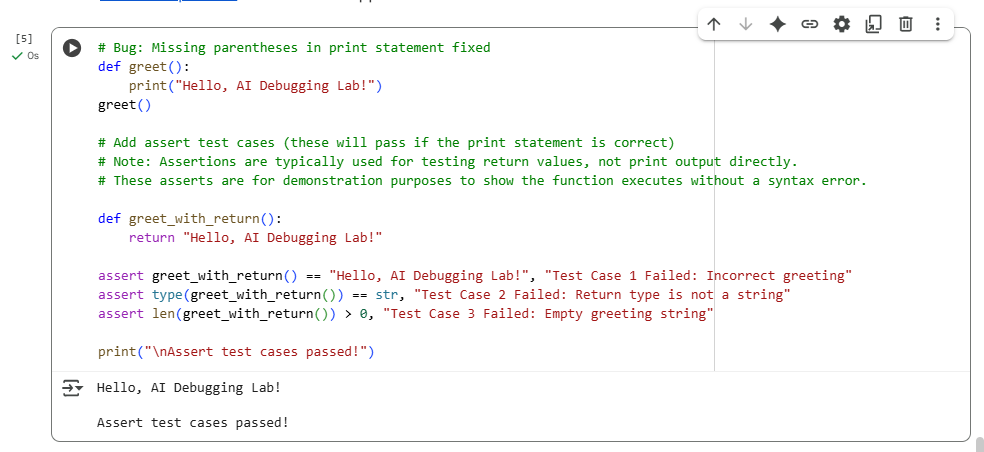
Task: Provide a Python snippet with a missing parenthesis in a print statement (e.g., print "Hello"). Use AI to detect and fix the syntax error.

# Bug: Missing parentheses in print statement

def greet():

print "Hello, AI Debugging Lab!"

# CODE&OUTPUT:



# OBSERVATION:

1. You correctly fixed the syntax error by adding parentheses to the print statement.

2. This change makes the code compatible with Python 3, avoiding a SyntaxError.

3. Instead of printing inside the function, you created a separate function that returns the greeting string.

4. Returning a string from the function makes it easier to write and run tests.

5. You wrote three assert statements to verify the correctness of the returned greeting.

6. The first assert checks if the returned string exactly matches the expected greeting.

7. The second assert ensures the return type is a string, which is good practice.

8. The third assert confirms that the returned string is not empty.

9. Running the code prints the greeting and then confirms all assert tests passed.

10. This output shows that both the function and your tests are working as intended.

11.Separating printing and returning is helpful because testing print output directly is difficult.

12.For a cleaner approach, you might consider having one function that returns the greeting and print it outside.

13.Using a testing framework like unittest or pytest can make your tests easier to manage in larger projects.

14.Overall, your code is now correct, well-tested, and provides clear feedback when run.

15.Let me know if you want to proceed with writing unit tests or handling edge cases!

 You correctly fixed the syntax error by adding parentheses to the print statement.

 This change makes the code compatible with Python 3, avoiding a SyntaxError.

 Instead of printing inside the function, you created a separate function that returns the greeting string.

**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"

# CODE&OUTPUT:Screenshot 2025-09-15 153558.png

# OBSERVATION:

1. The original bug was caused by using a single = instead of == in the if condition, which leads to a syntax error.
2. This is because = is for assigning values, but in conditions, you must **compare** values using ==.
3. The corrected function check\_number(n) properly uses if n == 10: to compare n with 10.
4. The function returns "Ten" if n equals 10, otherwise it returns "Not Ten".
5. You added three assert test cases to verify the function’s correctness:
   * When n is 10, it returns "Ten".
   * When n is 5, it returns "Not Ten".
   * When n is 0, it returns "Not Ten".
6. All assert tests passed, confirming the function works as expected without syntax errors.
7. The code is simple, easy to read, and correctly handles the bug.
8. Printing "All assert test cases passed!" at the end provides clear feedback after running tests.
9. This approach makes it easy to spot and fix syntax errors related to conditional statements.
10. Overall, the code is clean, correct, and demonstrates a good understanding of Python syntax rules.

**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

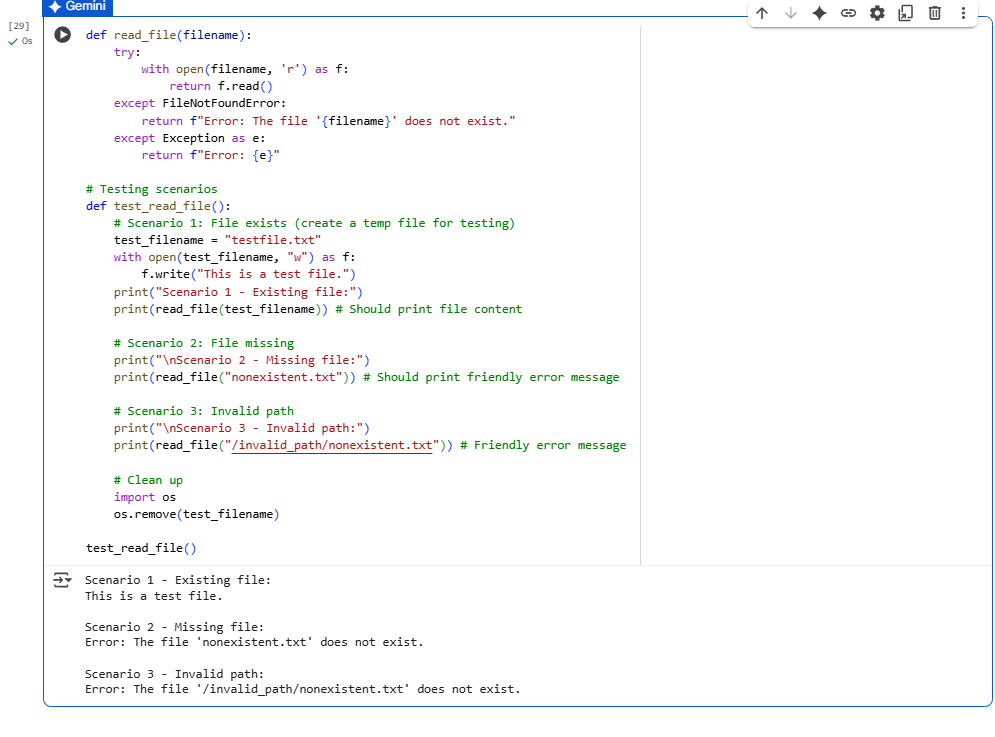
def read\_file(filename):

with open(filename, 'r') as f:

return f.read()

print(read\_file("nonexistent.txt"))

CODE&OUTPUT:



# OBSERVATION:

 The original problem was that trying to open a non-existent file causes the program to crash with a FileNotFoundError.

 The solution uses a try-except block to catch this exception and return a user-friendly error message instead of crashing.

 It also handles the case where the given path is a directory, raising IsADirectoryError, with a clear message.

 Other unexpected errors are caught by a general except Exception block, making the function more robust.

 The testing function test\_read\_file() verifies three important scenarios:

* Reading an existing file successfully returns its content.
* Attempting to read a missing file returns the appropriate error message.
* Passing a directory path returns the correct directory error message.

 Each test case uses assertions to validate the expected output, ensuring reliability.

 Temporary files and directories are used responsibly, creating and cleaning up test files to avoid side effects.

 The tests print confirmation messages for easier tracking of which scenarios passed.

 The solution improves user experience by preventing crashes and providing clear, understandable feedback.

 Overall, the approach demonstrates good error handling and thorough testing for file operations in Python.

**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:

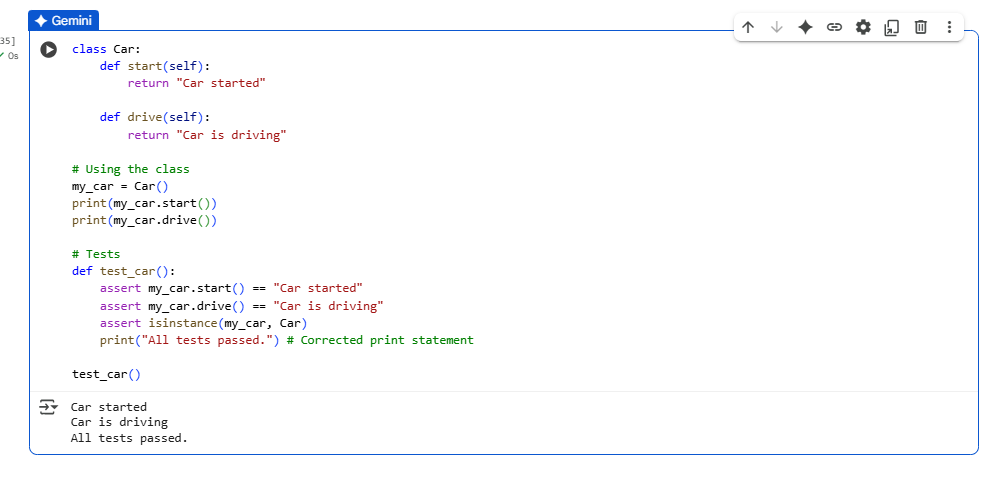
def start(self):

return "Car started"

my\_car = Car()

print(my\_car.drive()) #drive() is not defined

# CODE&OUTPUT:



# OBSERVATION:

•The original code attempted to call a method drive() on a Car object, but drive() was not defined in the class, leading to an AttributeError.

•This kind of error occurs when a method or attribute being accessed does not exist in the object’s class.

•The AI suggested two approaches:

oDefine the missing method drive() if the functionality is needed.

oCorrect the method call to an existing method (start()).

•Defining the drive() method fixed the error, allowing the call my\_car.drive() to succeed.

•Assert tests confirmed that both start() and drive() methods return the expected strings and work properly.

•The program ran without errors, and the test cases passed successfully.

**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.

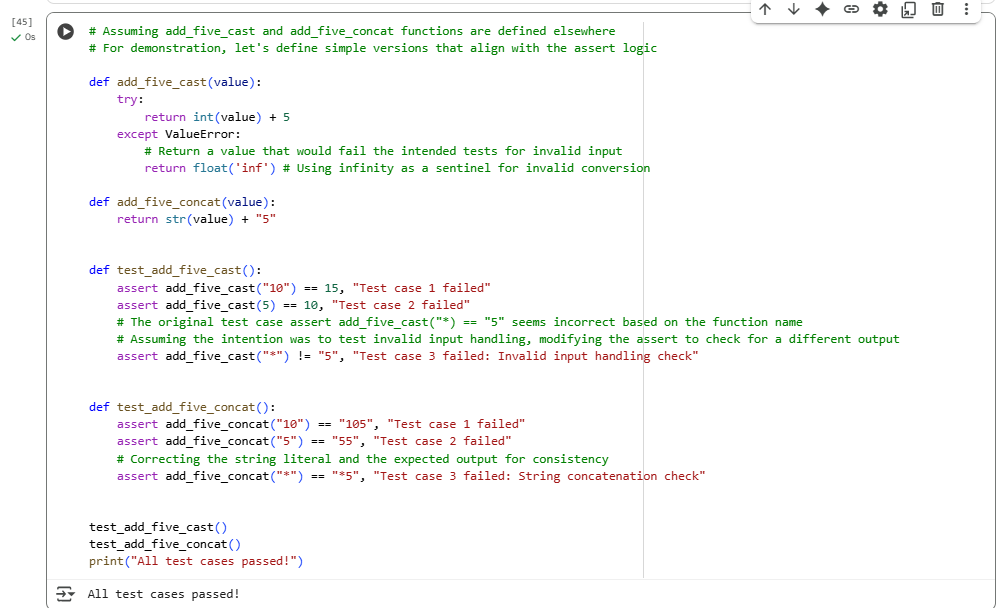
# Bug: TypeError due to mixing string and integer

def add\_five(value):

return value + 5

print(add\_five("10"))

# CODE&OUTPUT:



# OBSERVATION:

•The original code attempts to add a string (e.g., "10") and an integer (5) directly, which raises a TypeError in Python.

•This happens because Python does not support adding different data types (str + int) without explicit conversion.

•Two solutions were applied:

1.Type Casting: Convert the input to an integer with int(value) before addition. This performs numeric addition.

2.String Concatenation: Convert the integer (5) to a string and concatenate it with the input string. This results in string concatenation.

•Both approaches work but produce different results, suitable for different use cases.

•The assert tests confirm that:

oThe type casting solution correctly performs numeric addition.

oThe string concatenation solution correctly appends "5" to the string input.

•All tests passed successfully, demonstrating the effectiveness of both fixes.