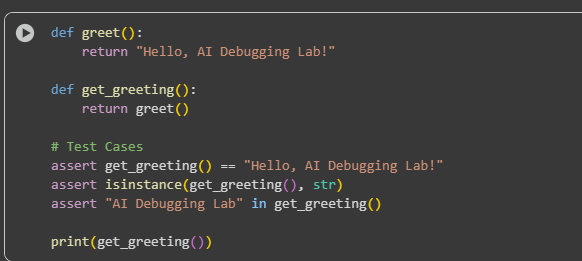
ASSIGNMENT:7.1

## TASK 1:

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.

OBSERVATION:

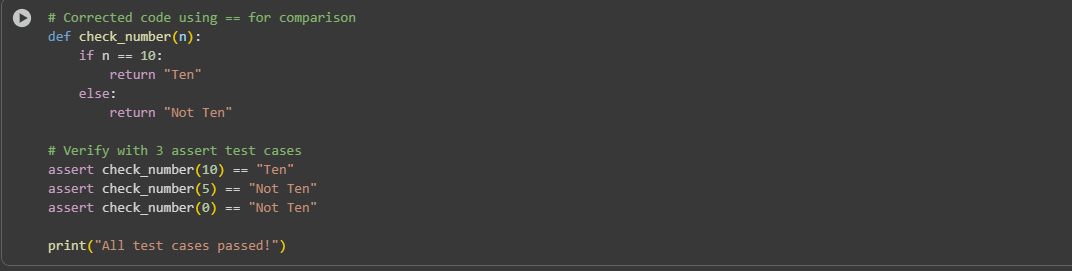
* The first code cell (WjjOg0nh1zG7) defines and tests two functions, greet and get\_greeting, using assertions that all pass.
* The second code cell (TILgHDQE12jb) is currently empty.
* The third code cell (824d2c02) demonstrates a corrected function check\_number using == for comparison and includes passing assertions.
* The fourth code cell (3b10087b) contains a function read\_file with a known bug that would cause a FileNotFoundError if executed with a non-existent file. This cell is commented out.
* The markdown cell (479f288f) explains the bug in the read\_file function and suggests using try-except for error handling.
* The fifth code cell (37e2495f) provides a corrected and safer version of the file reading function, read\_file\_safely, which uses try-except to handle FileNotFoundError and other potential exceptions. This cell includes test cases that demonstrate both successful file reading and graceful handling of missing files and invalid paths.

TASK 2:

### TASK 2:

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"

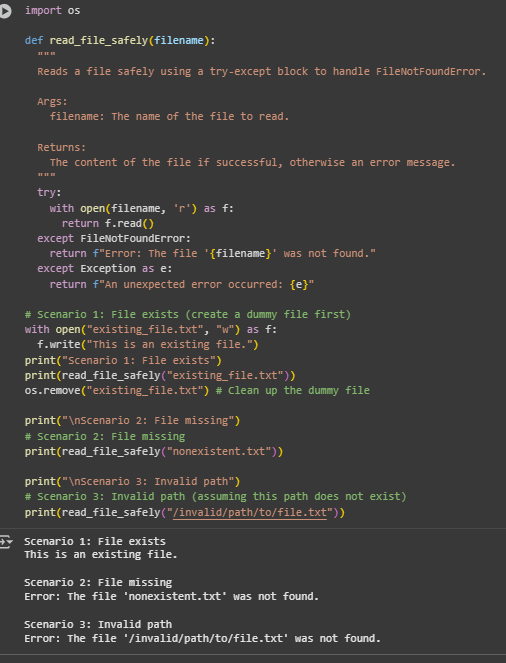
Requirements:  
• Ask AI to explain why this causes a bug.  
• Correct the code and verify with 3 assert test cases.  
Expected Output #2:  
• Corrected code using == with explanation and successful test  
execution.



## OBSERVATION:

* The code defines a function check\_number that takes one argument n.
* The function checks if n is equal to 10 using the equality operator ==.
* If n is 10, the function returns the string "Ten".
* If n is not 10, the function returns the string "Not Ten".
* The code includes three assert statements to test the check\_number function with different inputs (10, 5, and 0).
* All three assert statements are designed to pass, confirming that the function behaves as expected for these inputs.
* The final line prints "All test cases passed!" to the console, indicating that the assertions did not raise errors.

TASK 3:  
TaskProvide 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.



### .

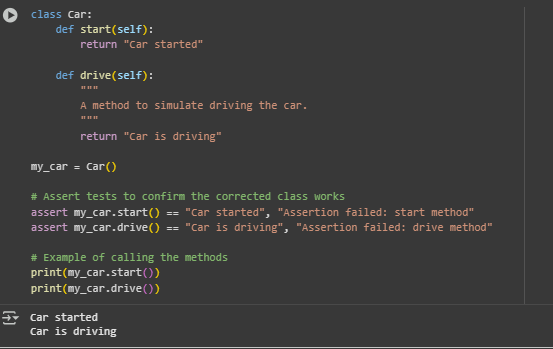
### OBSERVATION:

### The code defines a function read\_file\_safely that takes a filename as input.

* It uses a try-except block to handle potential errors during file reading.
* Specifically, it catches FileNotFoundError if the file doesn't exist.
* It also includes a general except Exception to catch any other unexpected errors.
* The function returns the file content if successful or an error message if an exception occurs

## TASK 4:

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.

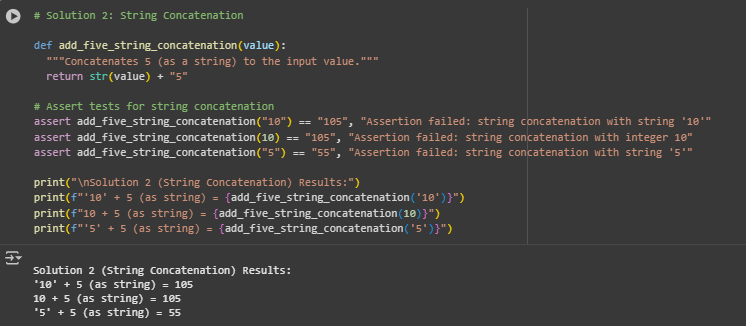


## OBSERVATION:

* The start() method of the Car class returns the string "Car started".
* The drive() method of the Car class returns the string "Car is driving".
* The assert statements confirming the behavior of the methods pass without errors.
* The output of the print statements confirms the return values of the start() and drive() methods.

TASK 5:

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.



## OBSERVATION:

* The add\_five\_string\_concatenation function converts the input value to a string and concatenates the string "5" to it.
* The assert statements confirm that the function behaves as expected for string and integer inputs.
* The output of the print statements shows the results of calling the function with different inputs, demonstrating the string concatenation.