**AI ASSISTED CODING**

**ASSIGNMENT-7.1**

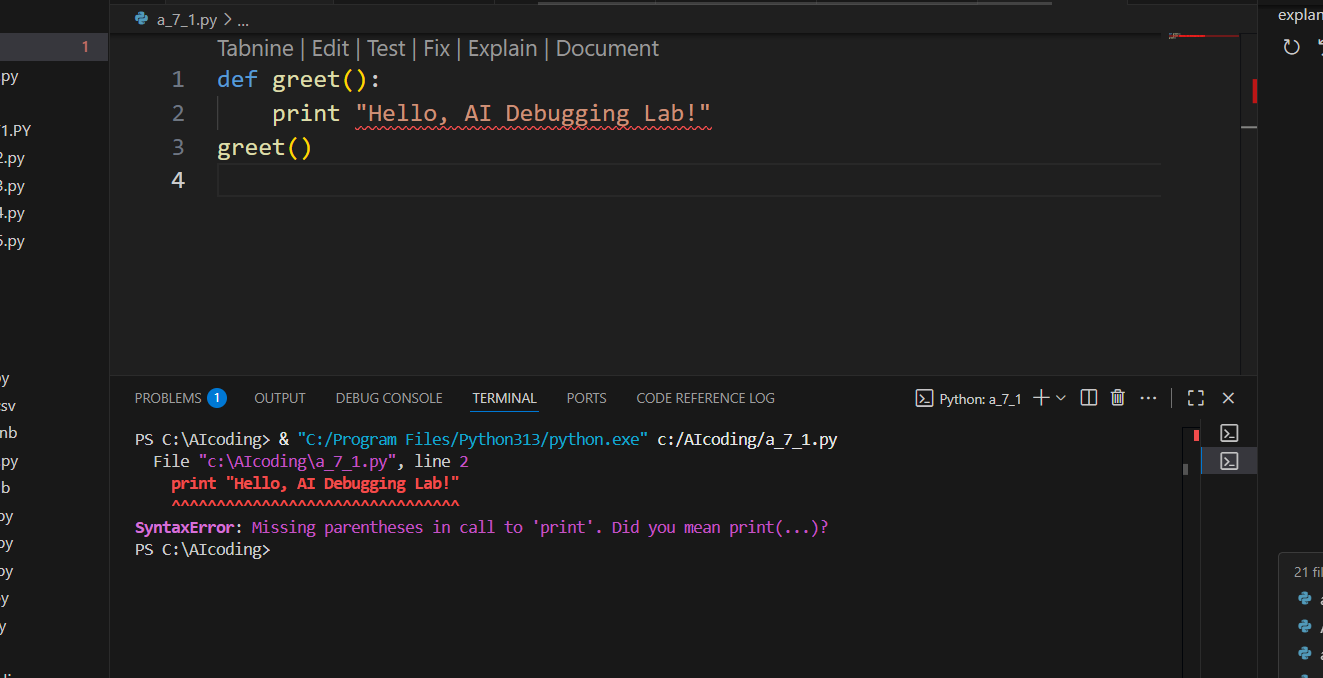
**2403A51260**

**SWAPNA MADISHETTI**

**BATCH-11**

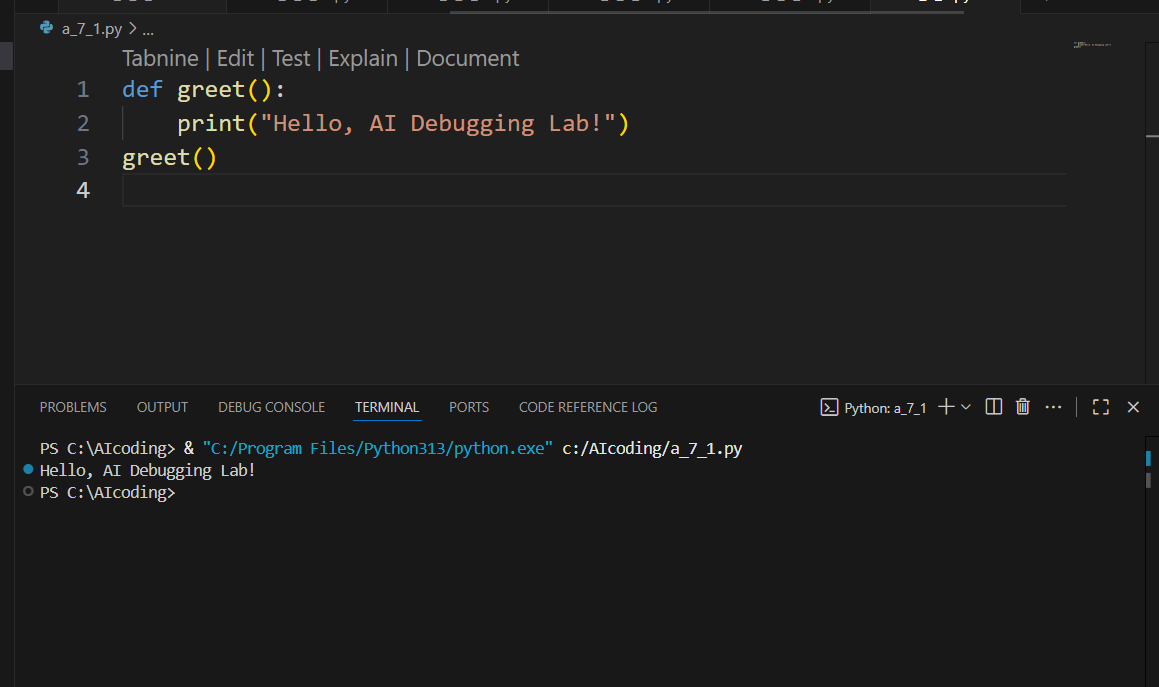
**TASK1:** (Syntax Errors – Missing Parentheses in Print  
Statement)  
 Provide a Python snippet with a missing parenthesis in a print  
statement (e.g., print "Hello"). Use AI to detect and fix the syntax error.  
# Bug: Missing parentheses in print statement  
def greet():  
 print "Hello, AI Debugging Lab!"  
greet()

ERRORED CODE:



**Prompt :** Detect and fix the syntax error in my code.

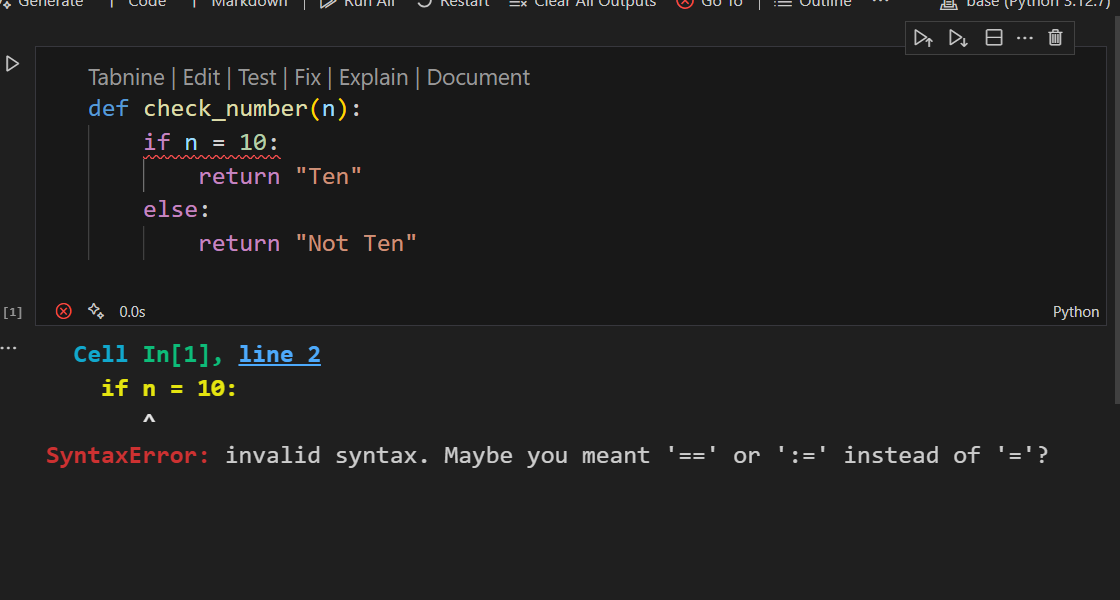
**Corrected code& output:**



**TASK-2:**(Logic Error – Incorrect Condition in an If  
Statement)  
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.

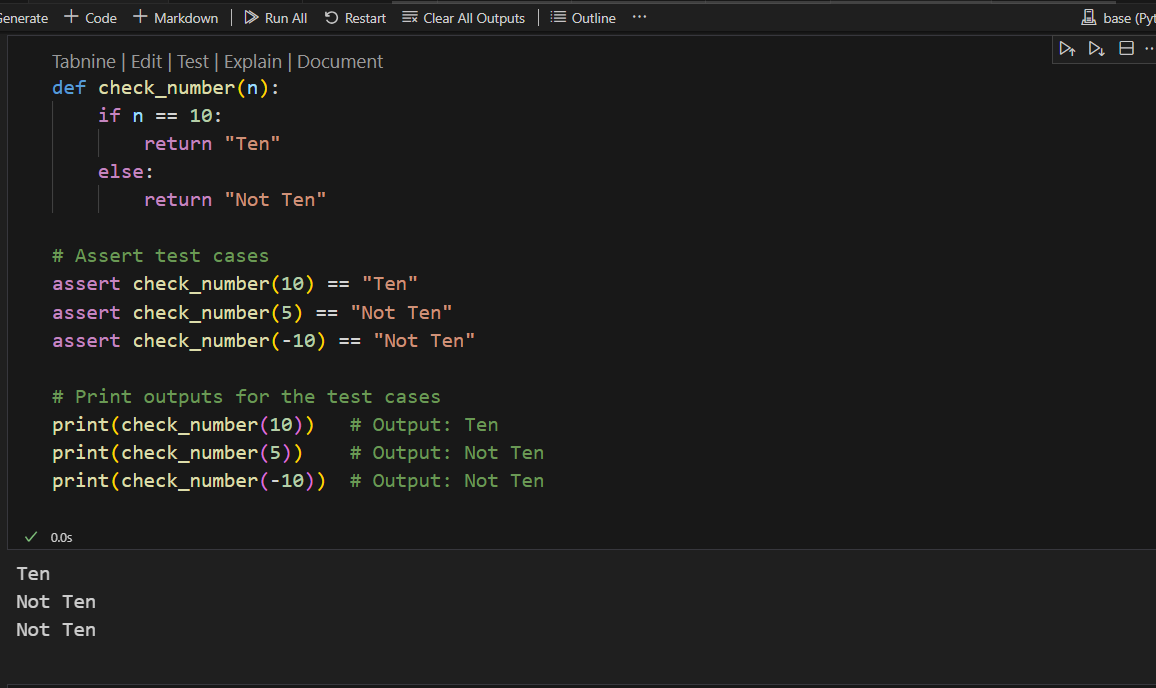
**ERRORED CODE:**



**Prompt given to AI:**

Correct the code and verify with three assert test cases.

**CORRECTED CODE:**

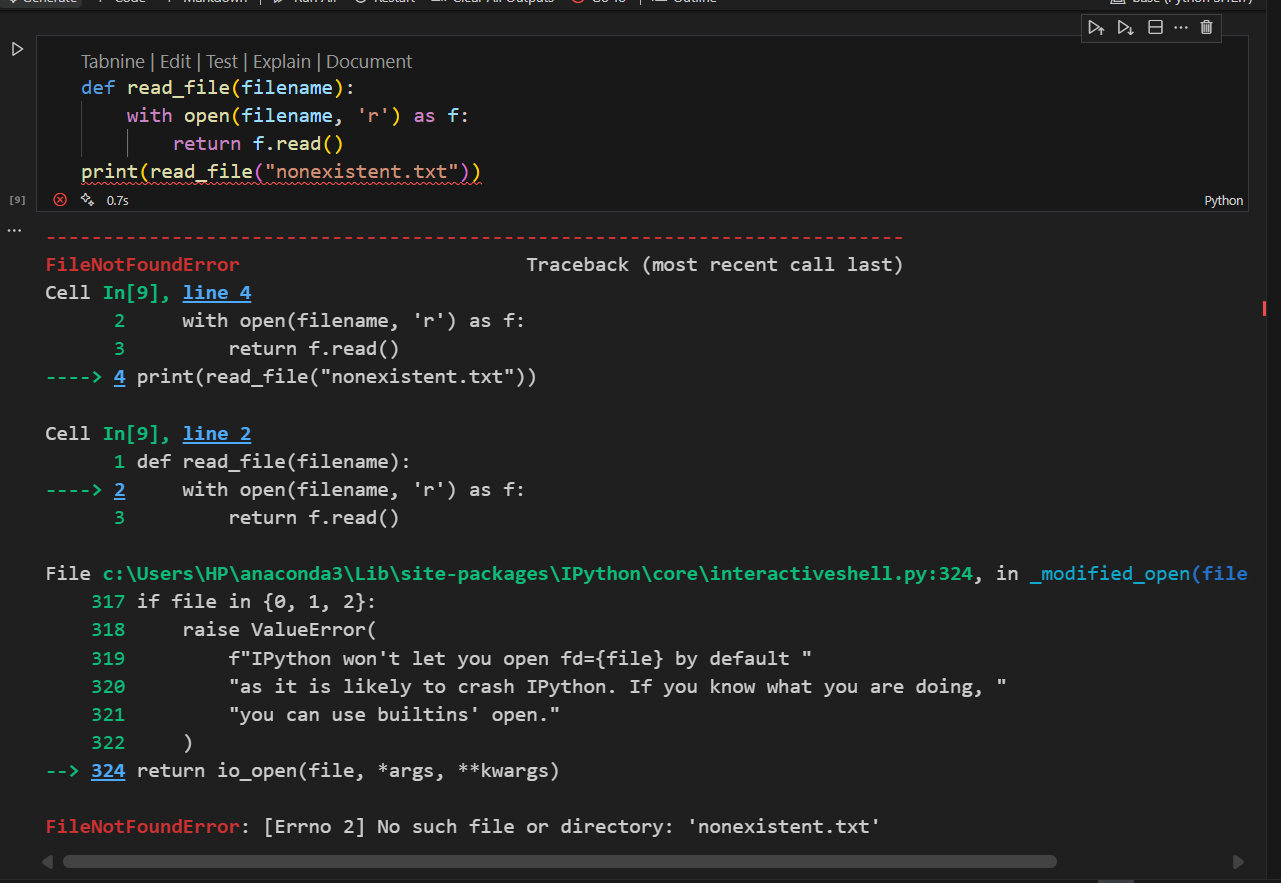


**Explanation for error:**

In Python, using a single equal sign = is an assignment operator, not a comparison operator.  
The if condition expects a Boolean expression, but n = 10 tries to assign the value 10 to n, which is invalid syntax inside a conditional expression.

**TASK-3:**(Runtime Error – File Not Found)  
 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

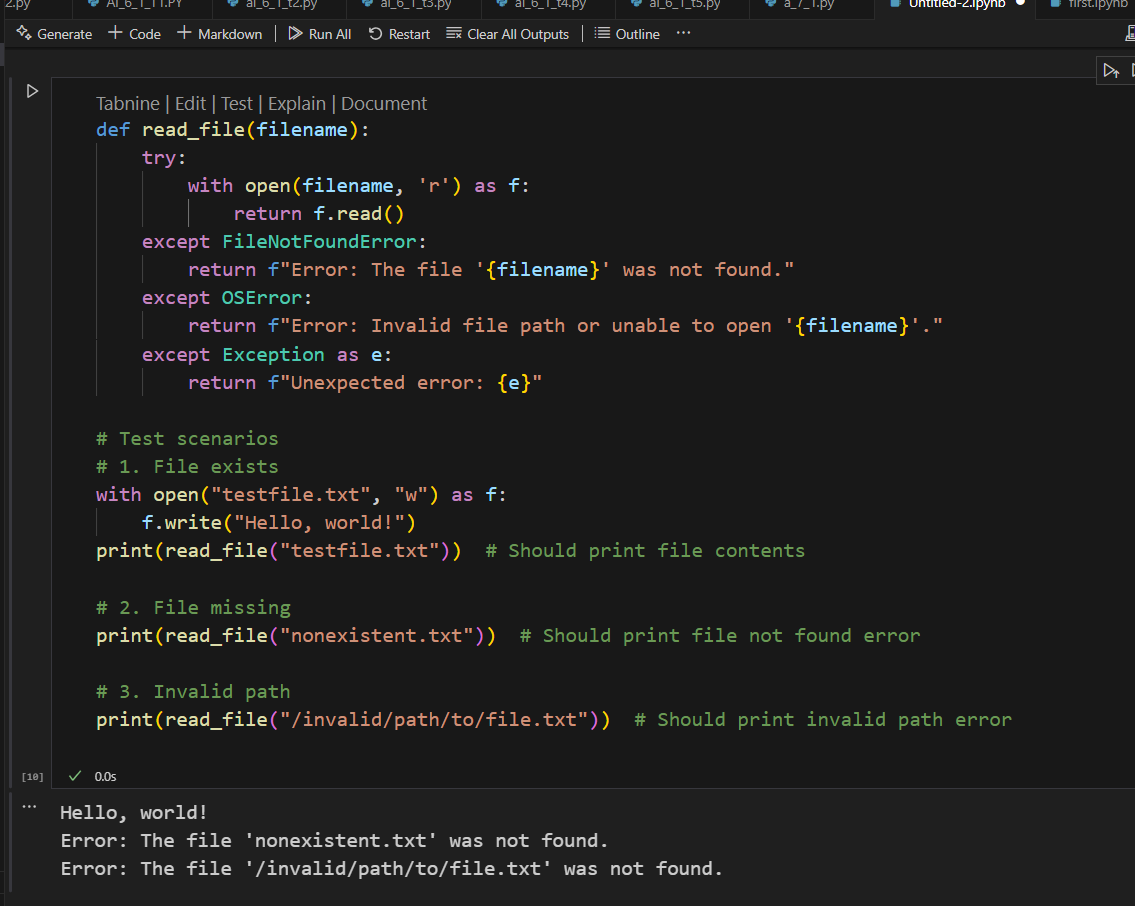
**CODE WITH ERROR:**

****

**Prompt:**

Implement a try-except block by Adding a user-friendly error messages Test with at least 3 scenarios: file exists, file missing, invalid  
path

**CORRECTED CODE & OUTPUT:**

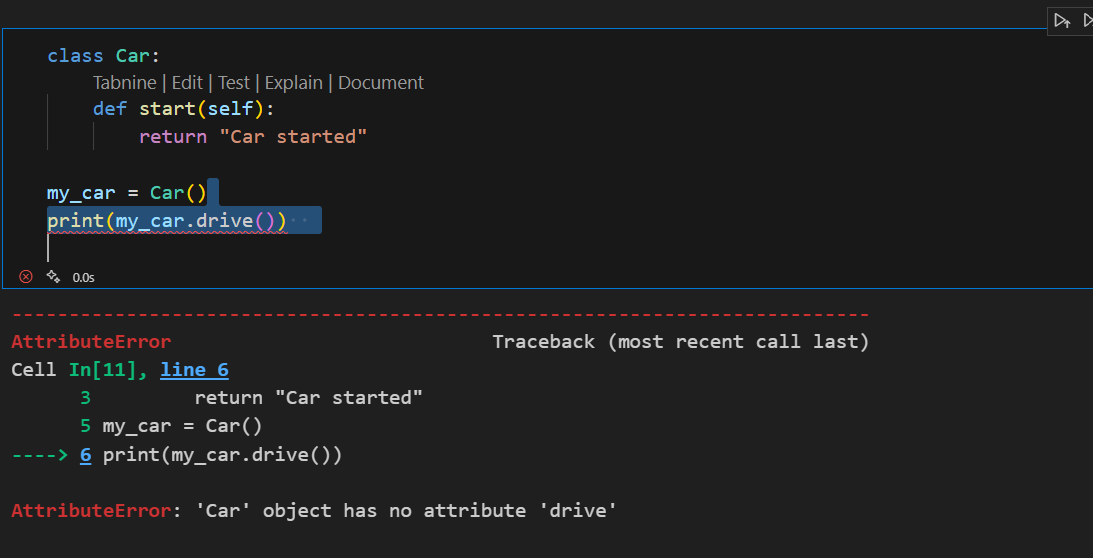


**TASK-4:(**AttributeError – Calling a Non-Existent Method)  
 Give a class where a non-existent method is called (e.g.,  
obj.undefined\_method()). Use AI to debug and fix.  
# Bug: Calling an undefined method

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.  
• Corrected class with clear AI explanation.

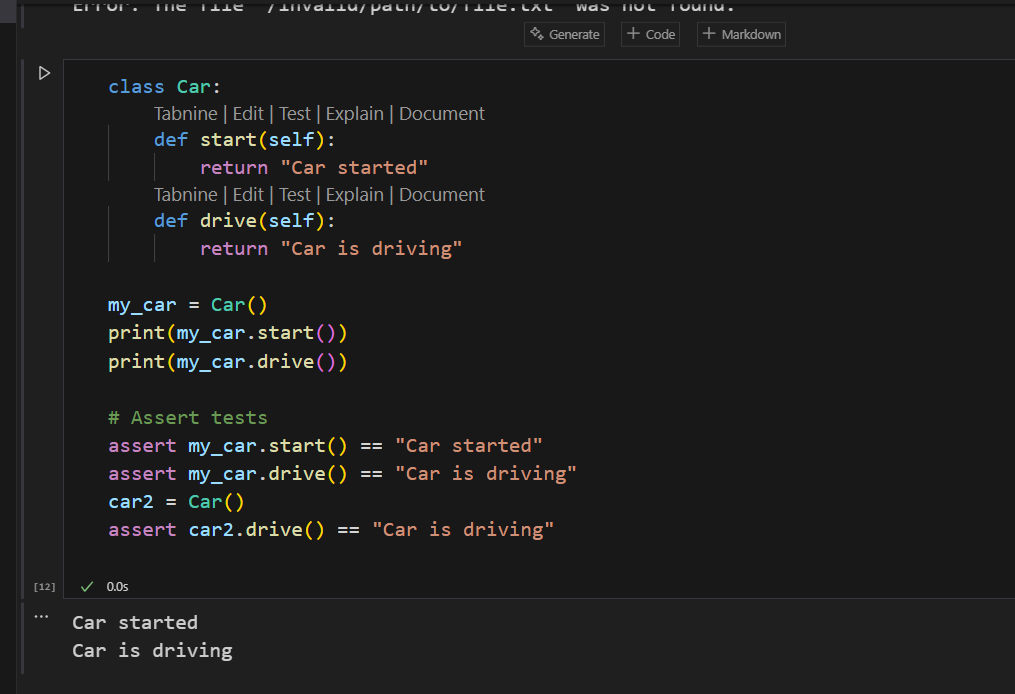
**CODE WITH ERROR:**



**PROMPT:**

correct the code and Use 3 assert tests to confirm the corrected class works.

**CORRECTED CODE WITH OUTPUT:**



**EXPLANATION:**

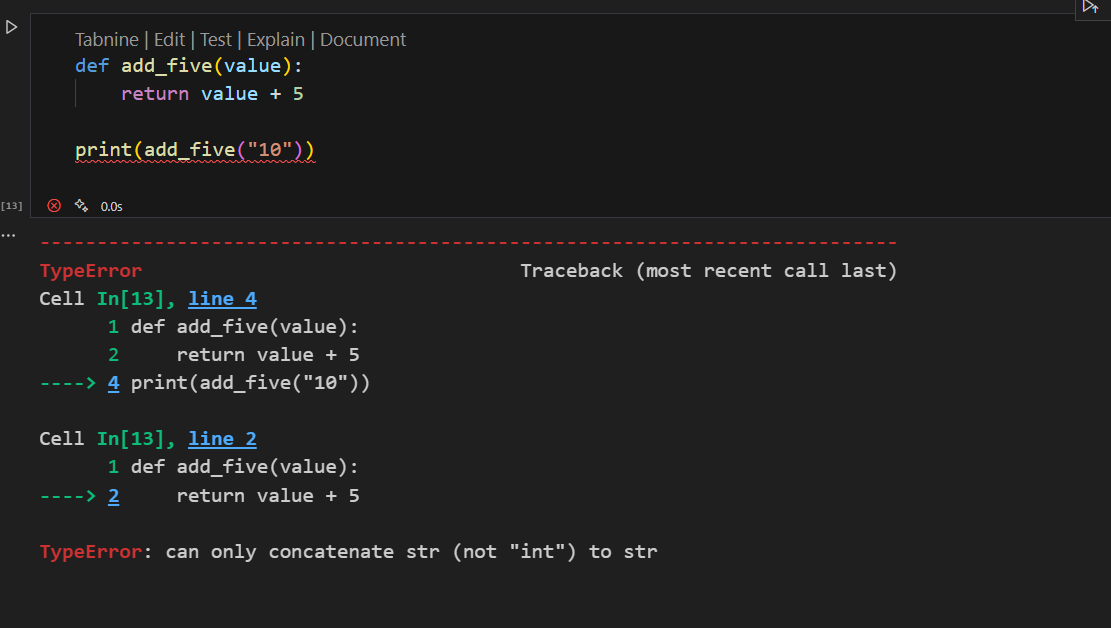
The error occurs because the Car class defines a method named start(), but later the code tries to call my\_car.drive().  
However, there is **no method named drive() defined in the class**.  
Now we must decide:

* Was the intent to call start()?
* Or should a new drive() method exist in the class?

Assuming **both functionalities are expected** (starting and driving a car), the correct approach is to **define the missing drive() method** in the class.

**TASK-5:**(TypeError – Mixing Strings and Integers in  
Addition)  
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

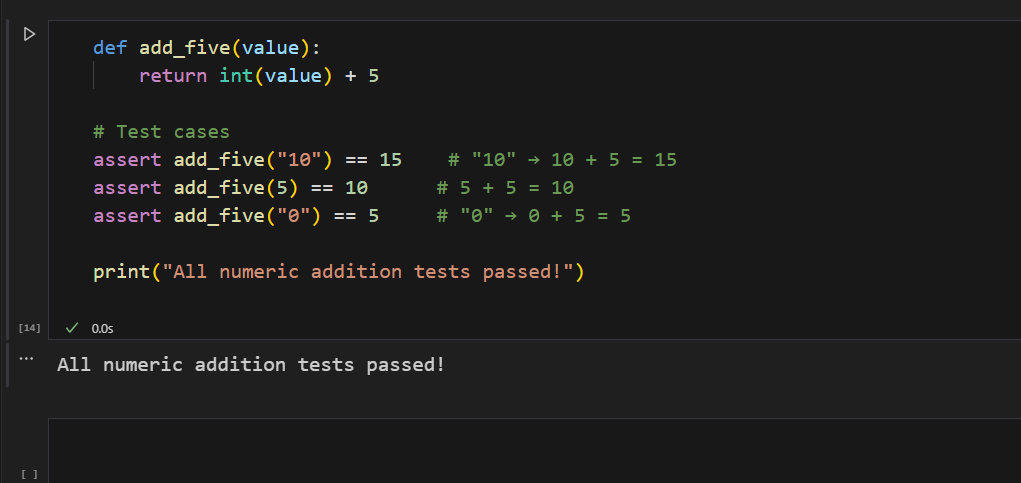
**CODE WITH ERROR:**

****

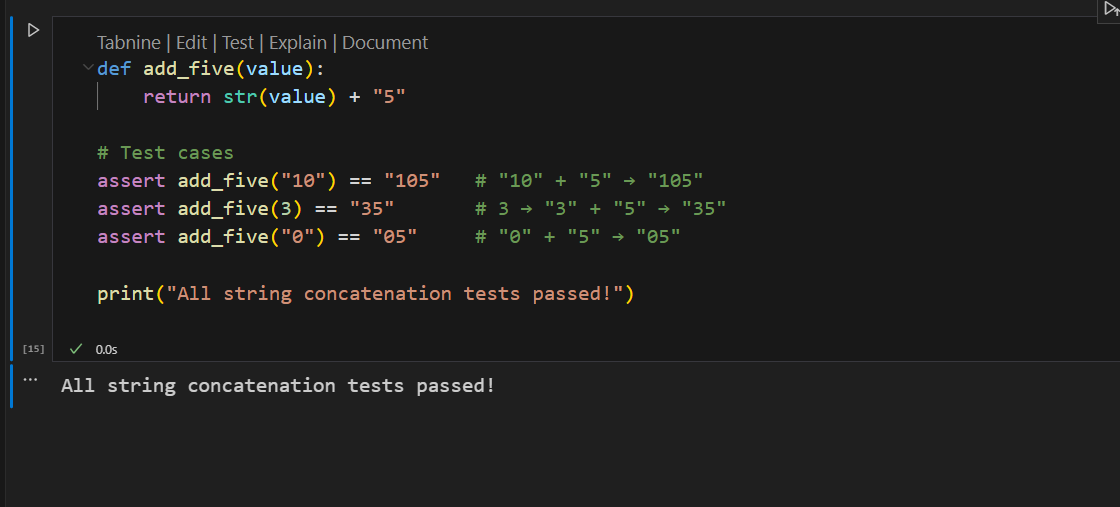
**CORRECTED CODES:**

**SOLUTION-1**:(Numerical addition)

**PROMPT:**

****

**SOLUTION-2:** (String concatenation)

****