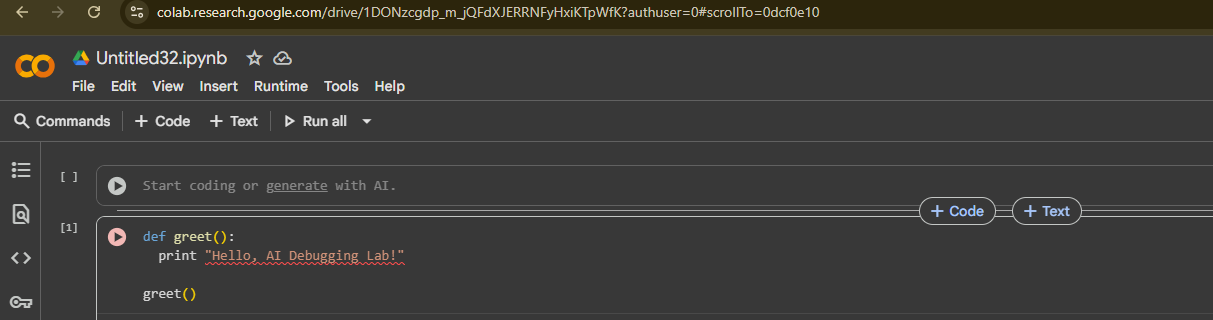
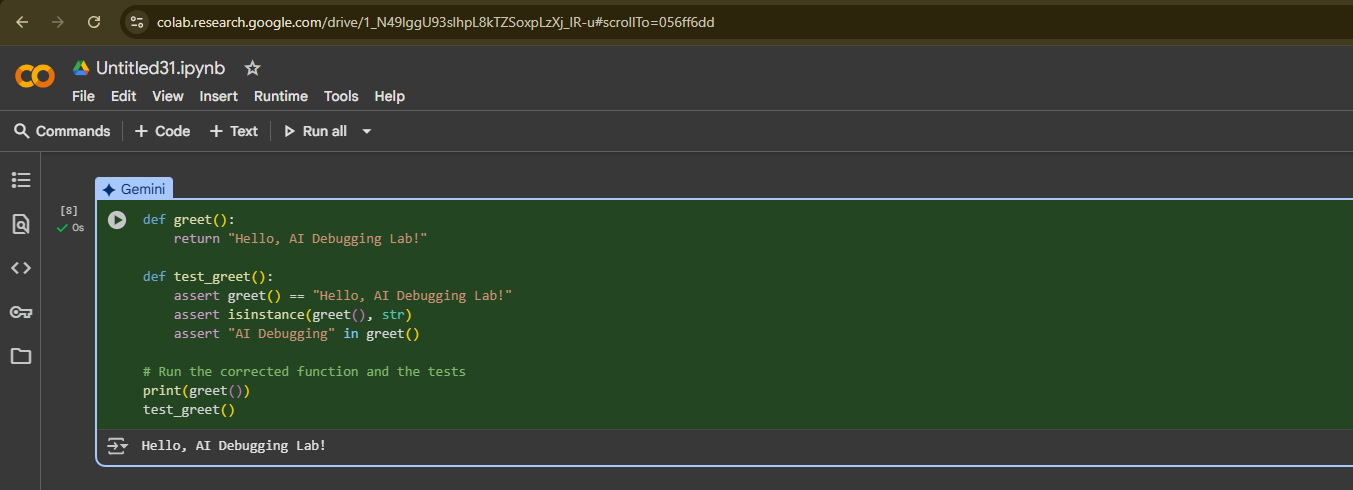
ASSIGNMENT-7.1

# Task-1:

Provide a Python snippet with a missing parenthesis in a print statement (e.g., print "Hello"). 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 should be: Corrected code with proper syntax and AI explanation

Code:



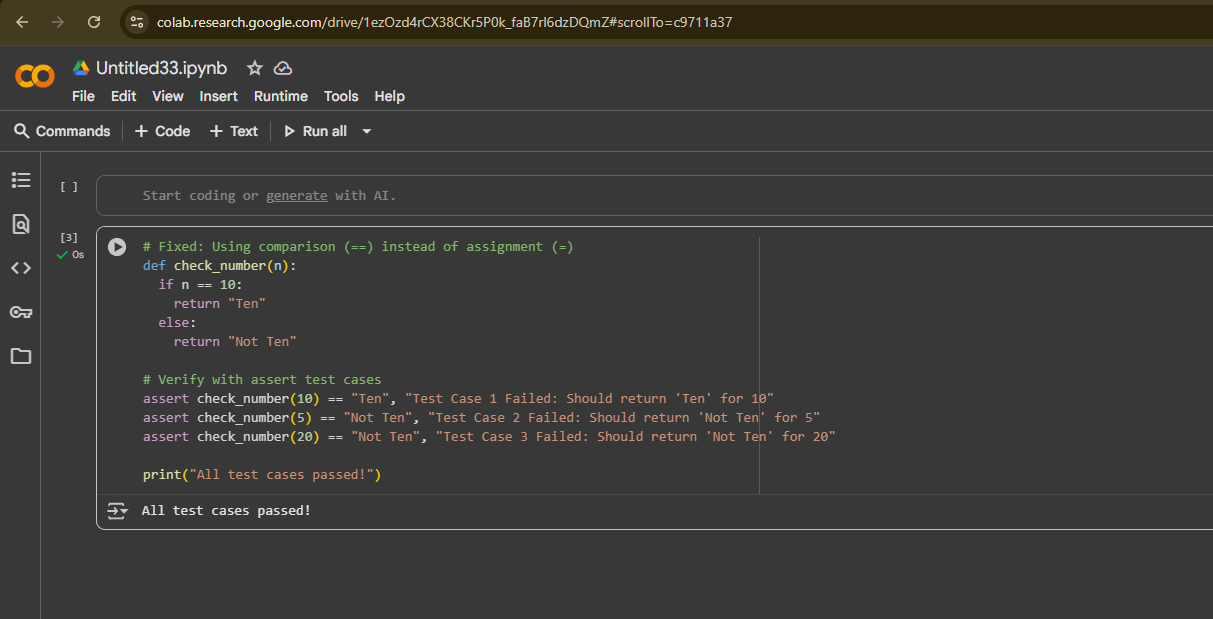


# Observation:

the key observation is about a **syntax difference between Python 2 and Python 3**. In Python 2, print was treated as a statement and did not require parentheses. However, in Python 3, print  is a function, and you must use parentheses around the arguments you are printing, like print(…) . The error you see is a direct result of using Python 2 syntax in a Python 3 environment.

# Task-2: 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.

Code:

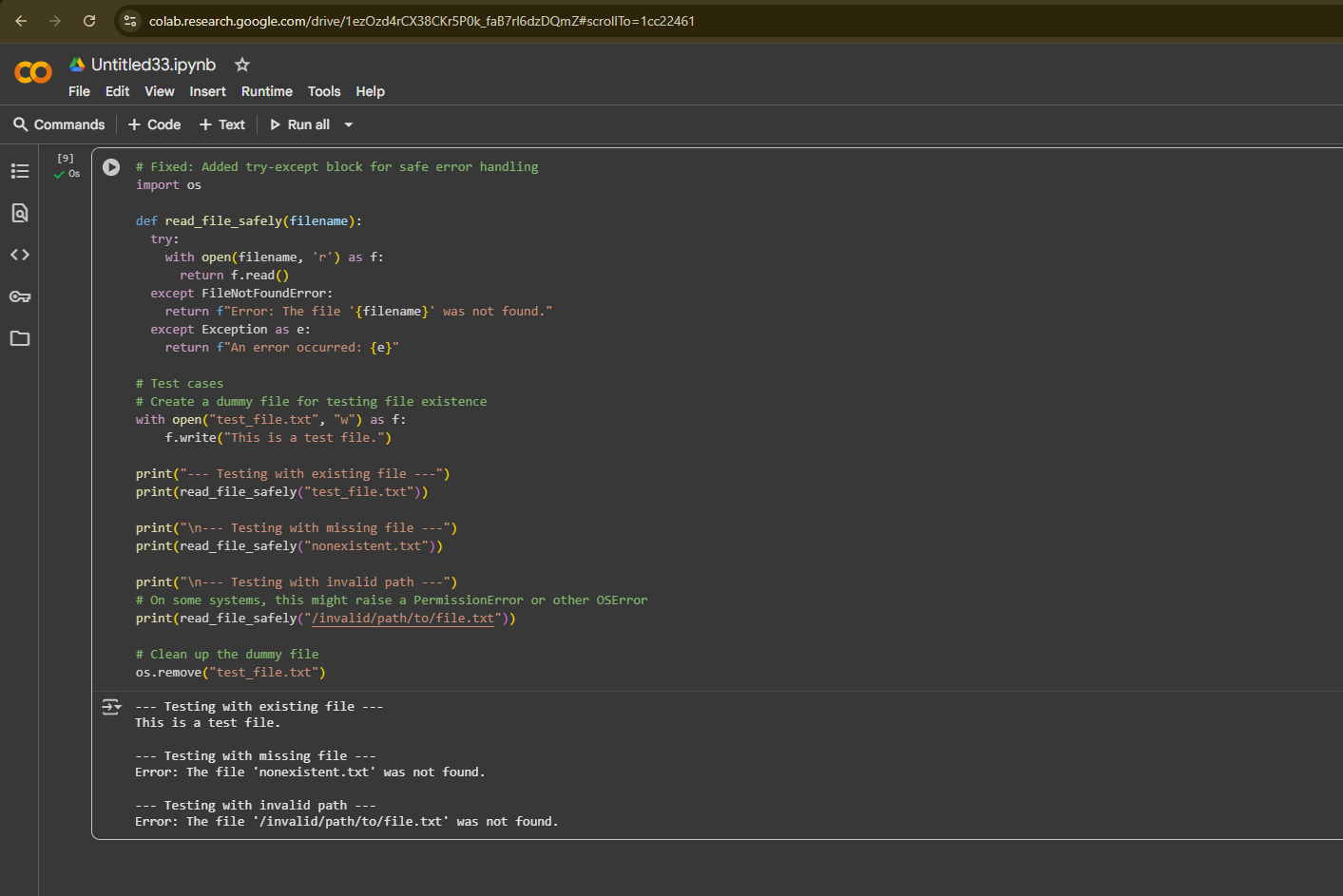


# Observation:

a crucial observation is the difference between the **assignment operator (=)** and the **equality comparison operator (==)** in Python. Using =  within an if statement (as was the original bug) is a common mistake that can lead to a SyntaxError or unintended behavior. The corrected code correctly uses == to check if the value of n is equal to 10. This highlights the importance of using the correct operator for the intended operation, whether it's assigning a value or comparing two values for equality.

# Task-3:

# 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. Output:



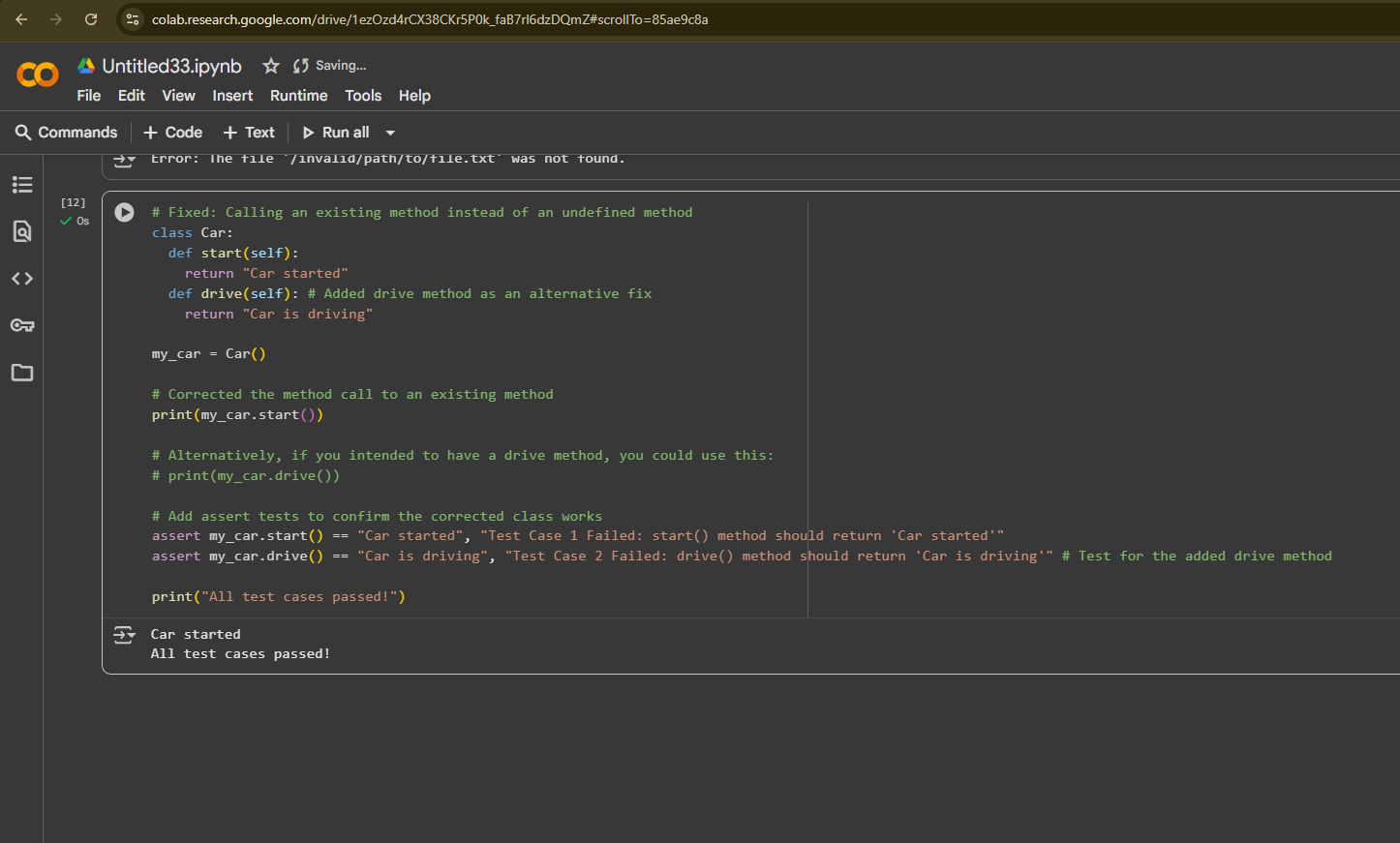
# Observation:

 a key observation is the importance of **robust error handling** when interacting with external resources like files. Attempting to open a file that doesn't exist or is in an inaccessible location can cause your program to crash with errors like FileNotFoundError. The try-except block effectively prevents this by "catching" potential errors and allowing you to handle them gracefully, providing informative messages to the user instead of abruptly stopping the program. This makes the code more reliable and user-friendly.

# Task-4:

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.

Output:



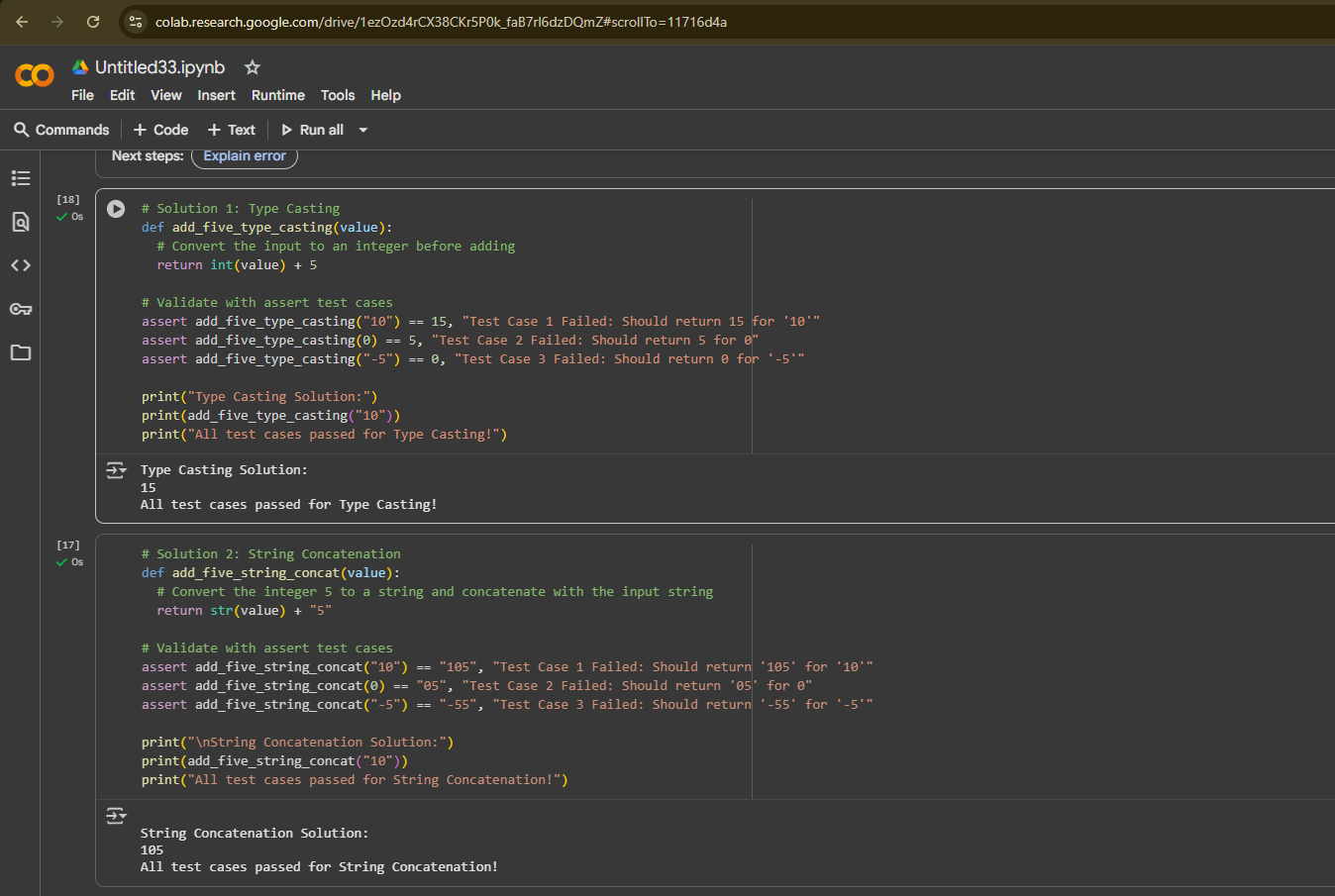
# Observation:

an important observation is related to **method calls on objects**. If you try to call a method that is not defined within a class (like the original attempt to call my\_car.drive, it will result in an Attribute Error The corrected code demonstrates how to fix this by either calling an existing method (my\_car.start()) or by adding the missing method definition (def drive(self):...) to the class. This highlights the need to ensure that the methods you call on an object are actually part of the object's class definition.

# 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 should be: • Corrected code that runs successfully for multiple inputs

Output:



# Observation:

The main observation is the importance of **data types** in programming and how attempting to perform operations (like addition) on incompatible data types can lead to errors such as TypeError. The code demonstrates this by trying to add a string and an integer directly. The solutions provided highlight that you need to ensure the data types are compatible for the desired operation, either by converting the string to an integer (type casting for arithmetic addition) or converting the integer to a string (string concatenation for combining text).