HTNO:2403A51286

ASSIGNMENT:7.1

Q1:

**TASK#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!"  
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.

**Prompt:**

(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()

**Requirements:**

• Run the given code to observe the error.

• Apply AI suggestions to correct the syntax.

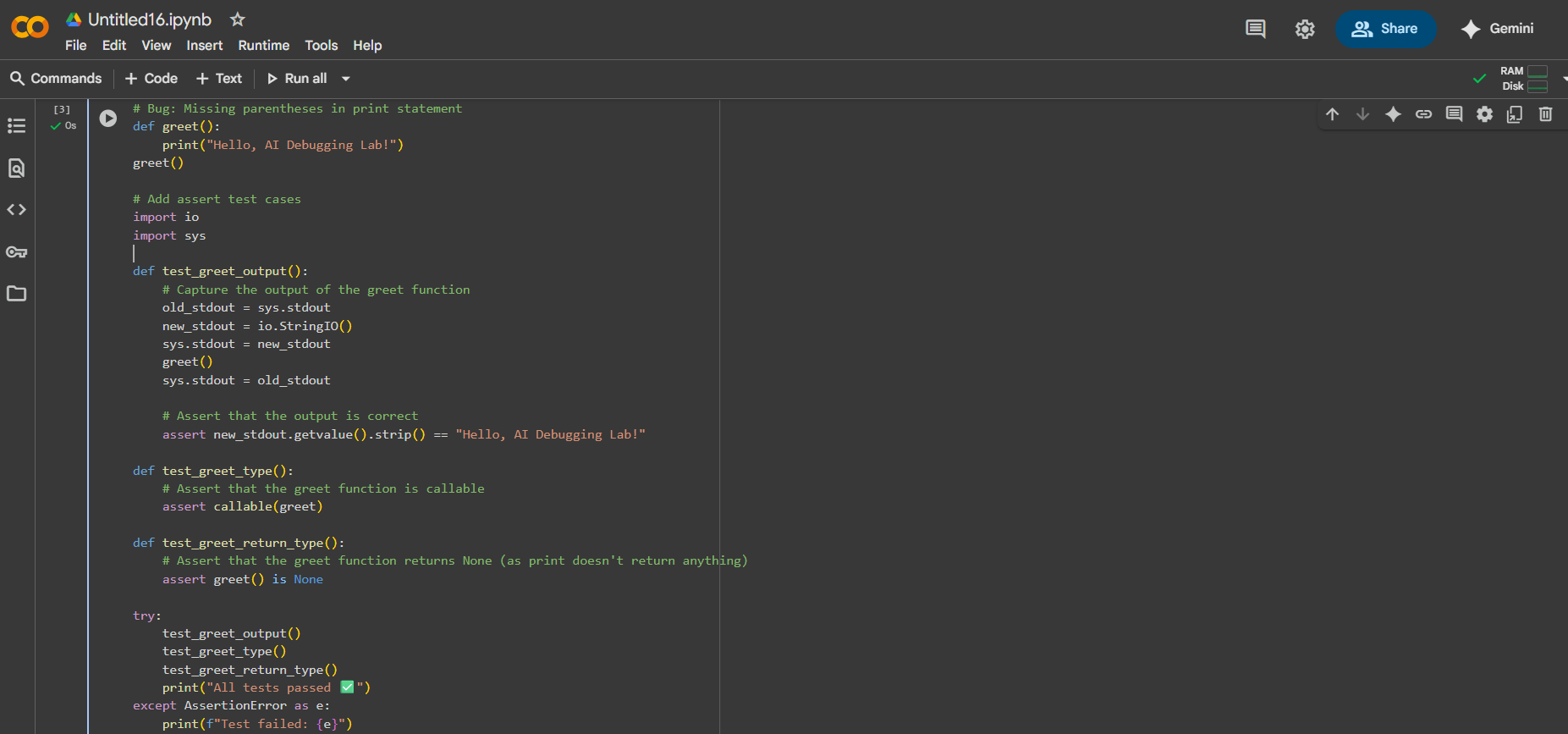
• Use at least 3 assert test cases to confirm the corrected code works.

**Expected Output #1:**

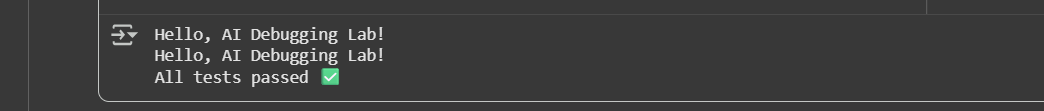
• Corrected code with proper syntax and AI explanation.

• Test cases showing “All tests passed ”.

**Code:**



Output:



**Task#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"

**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.

**Prompt:**

(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:

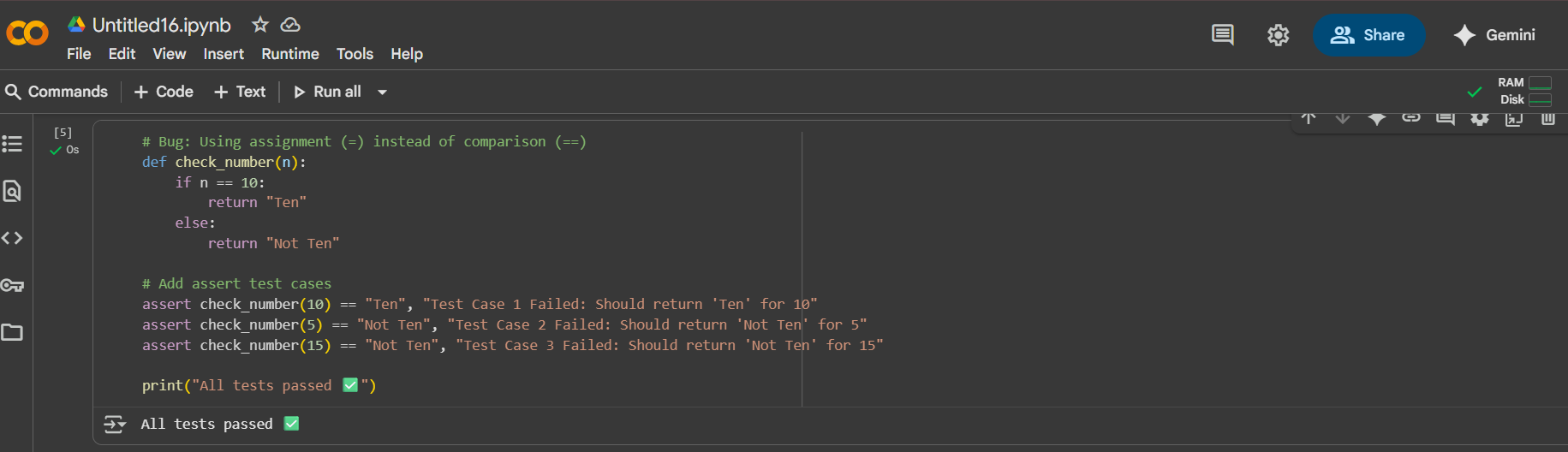
return "Not Ten"

**Requirements:**

• Ask AI to explain why this causes a bug.

• Correct the code and verify with 3 assert test cases.

**Code and Output:**



**Task#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"))

**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, invalidpath.

**Expected Output #3:**

• Safe file handling with exception management

**Prompt:**

(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"))

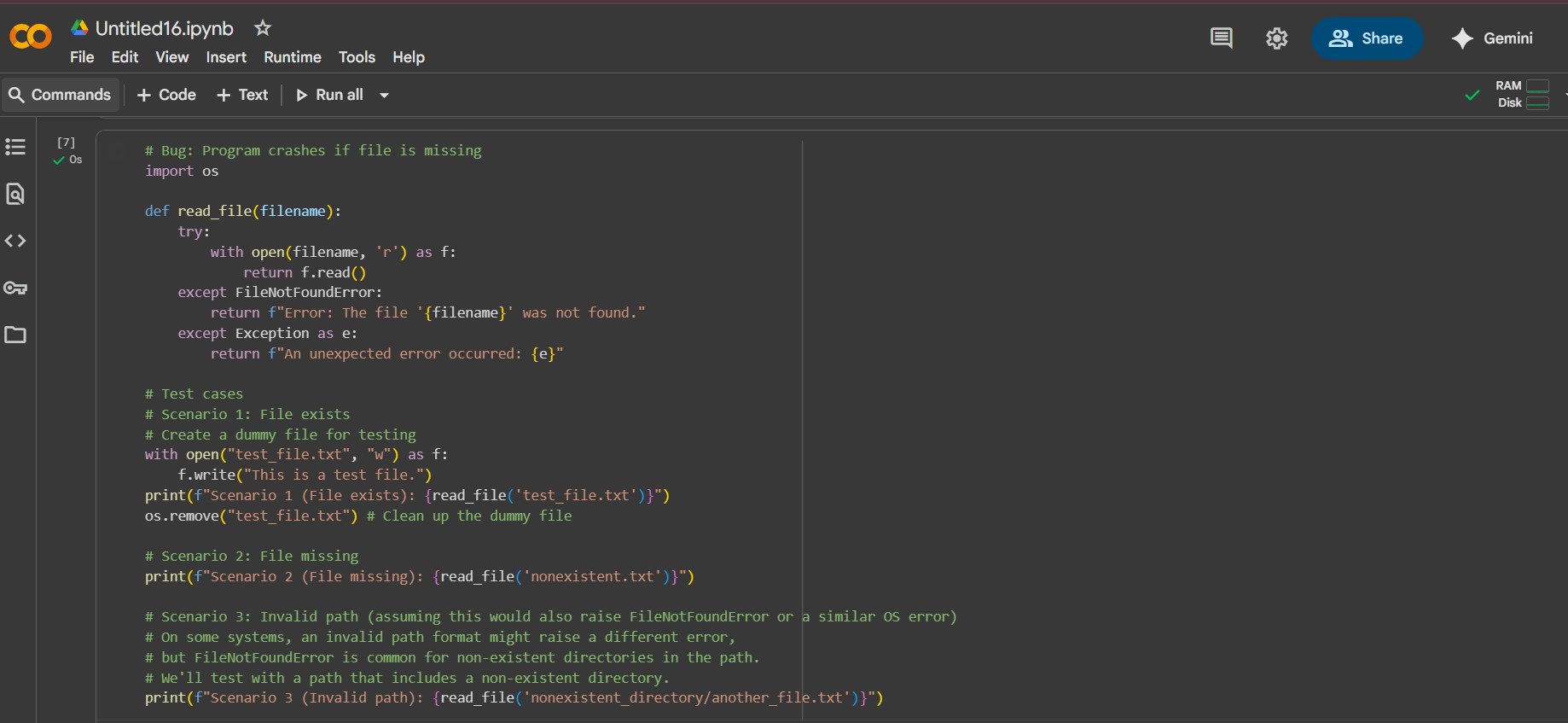
**Requirements:**

• Implement a try-except block suggested by AI.

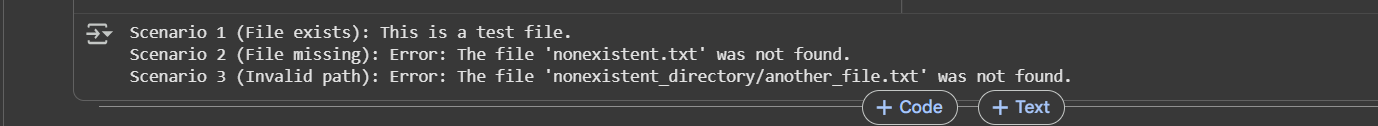
• Add a user-friendly error message.

• Test with at least 3 scenarios: file exists, file missing, invalid path.

**Code:**



**Output:**



**Task#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

**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.

**Prompt:**

(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()

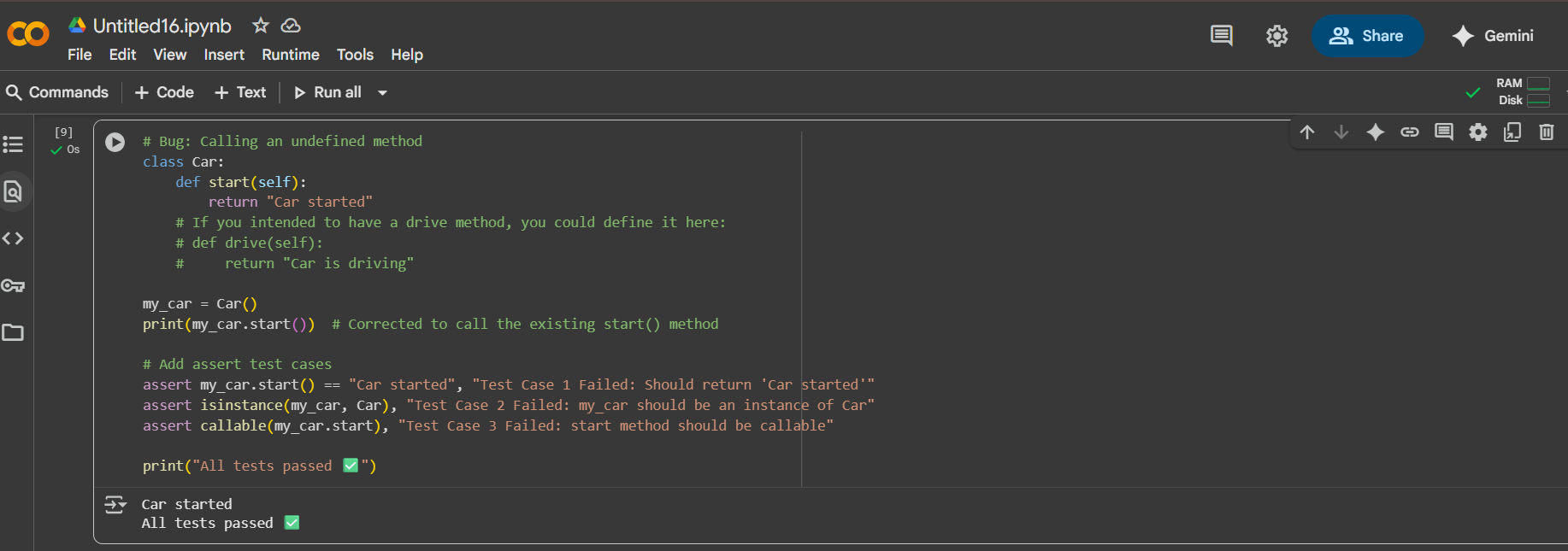
print(my\_car.drive()) # drive() is not defined

**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.

**Code and Output:**

****

**Task#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"))

**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

**Prompt:**

(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.

**Expected Output #5:**

• Corrected code that runs successfully for multiple cases.  
  
**Code and Output:**

