Assignment: 7.1

2403A51329

BANDI SHASHI KUMAR

**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():

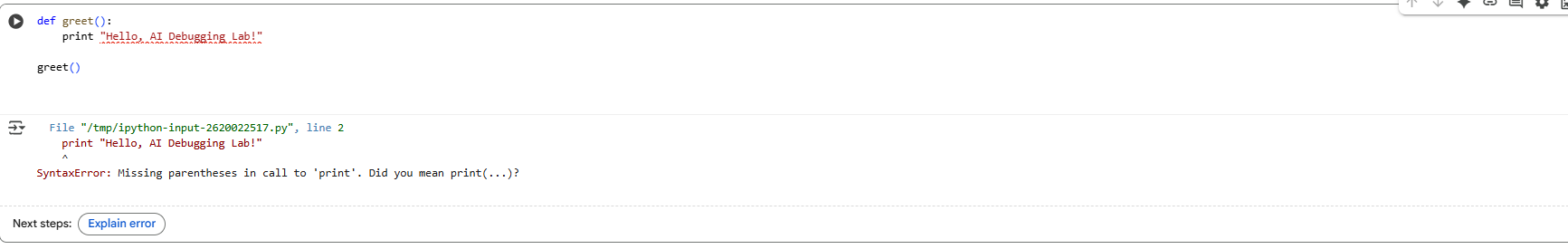
print "Hello, AI Debugging Lab!"

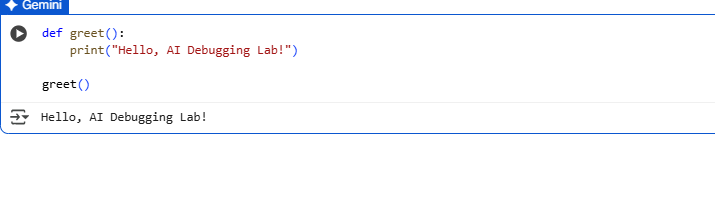
greet()

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.

**OUTPUT:**

****

****

**Observation – Task 1:**  
In this task, I learned how a small syntax mistake, such as missing parentheses in the print statement, can cause a program to fail in Python 3. By using AI-assisted debugging, I was able to quickly identify the error and correct it. The AI explanation helped me understand the difference between Python 2 and Python 3 syntax. Writing assert test cases further improved my confidence in verifying program correctness. This exercise highlighted the importance of careful debugging and reinforced how AI tools can simplify error detection and correction.

**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"

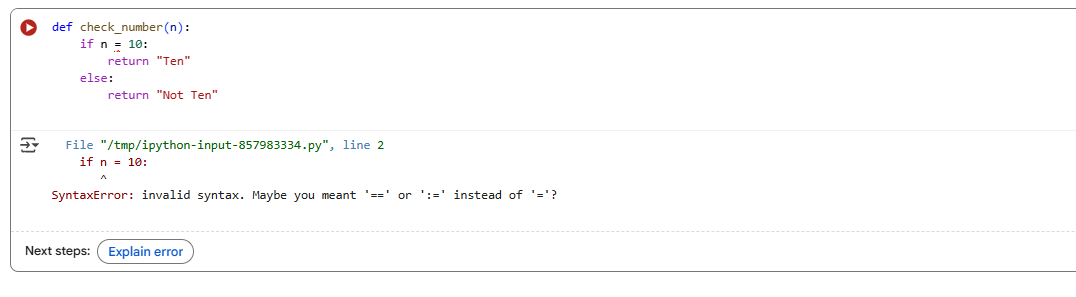
else:

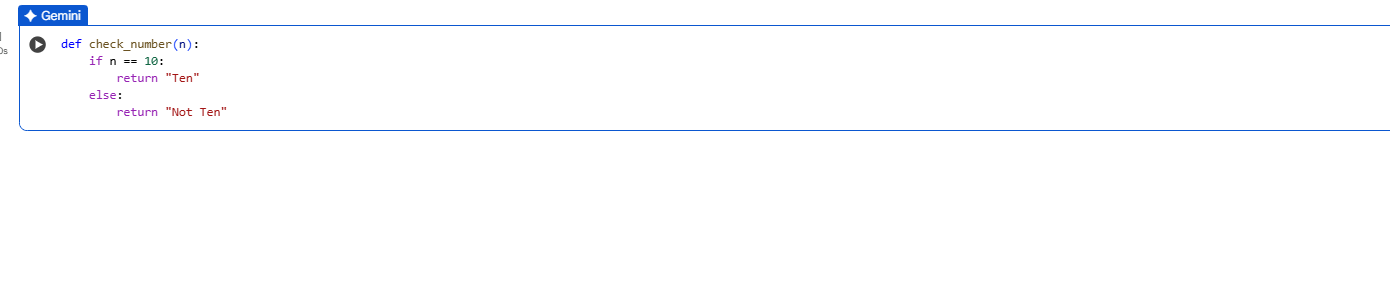
return "Not Ten"

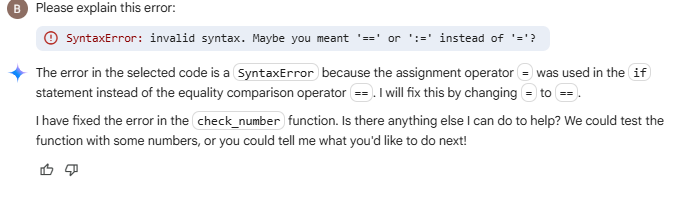
Requirements:

* Ask AI to explain why this causes a bug.
* Correct the code and verify with 3 assert test cases.

**OUTPUT:**

****

****

****

**Observation – Task 2:**  
In this task, I understood how using the assignment operator = instead of the comparison operator == leads to a logic error in Python. While = assigns a value, == is used to compare values. The AI tool quickly identified this mistake, explained the difference, and suggested the correction. After fixing the condition and adding assert test cases, the function worked correctly. This task reinforced the importance of carefully distinguishing between assignment and comparison operators and showed how AI can simplify detecting such subtle but critical programming errors.

**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):

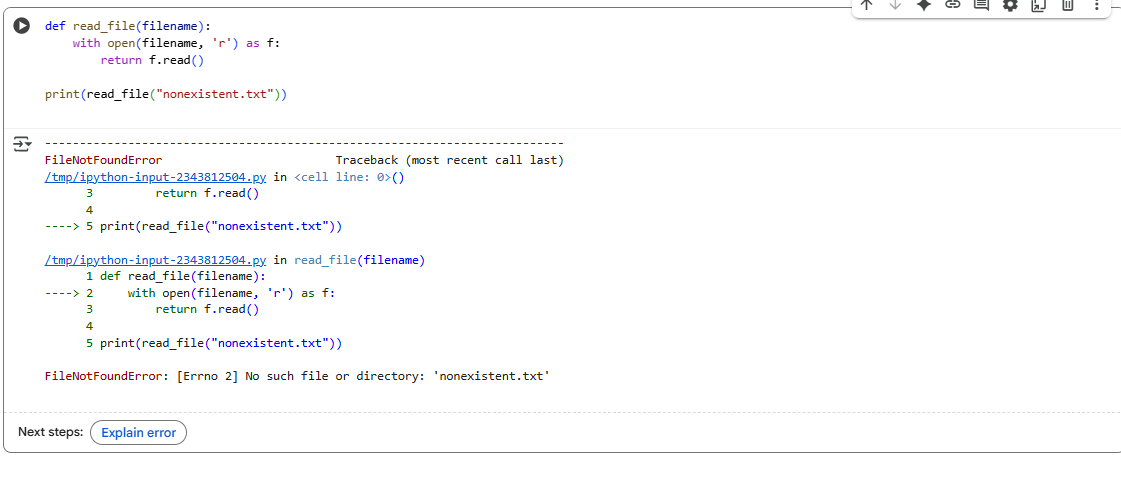
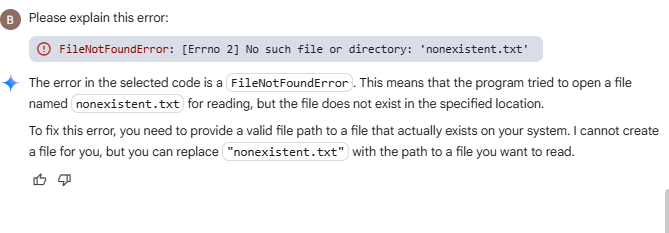
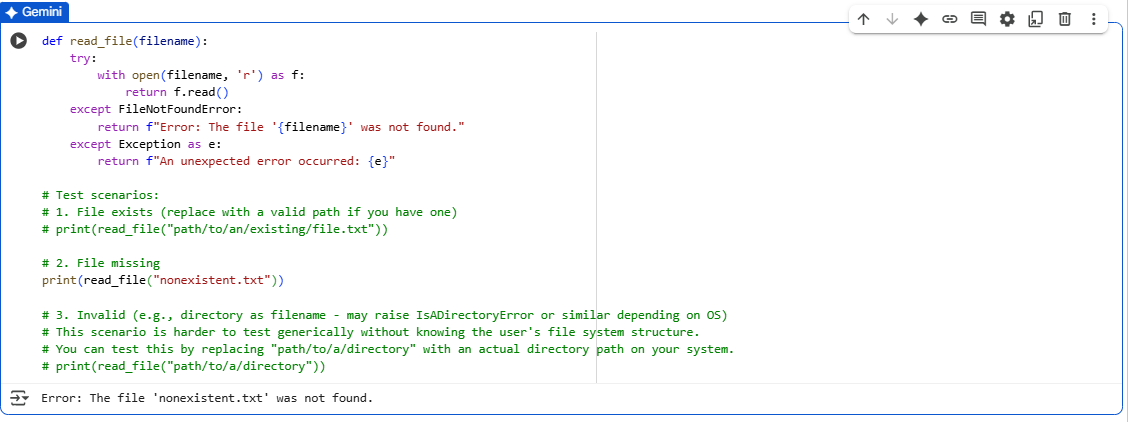
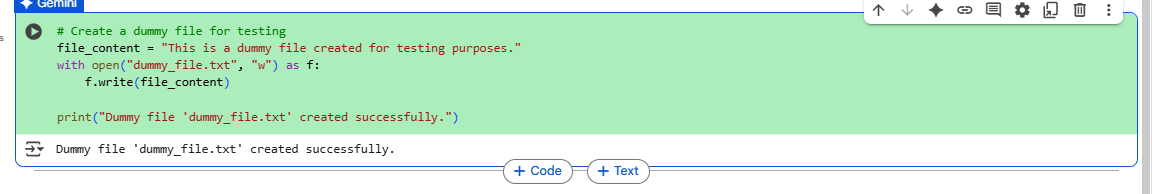
with open(filename, 'r') as f:

return f.read()

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

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.



**Observation – Task 3:**  
In this task, I learned how runtime errors like FileNotFoundError occur when a program tries to open a file that does not exist. Using AI-assisted debugging, I was able to apply a try-except block to handle such cases gracefully. The AI explained how to provide user-friendly error messages instead of letting the program crash. By testing different scenarios, I observed how exception handling improves reliability. This task highlighted the importance of safe file handling in real-world applications and how AI can guide effective error management practices.

**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"

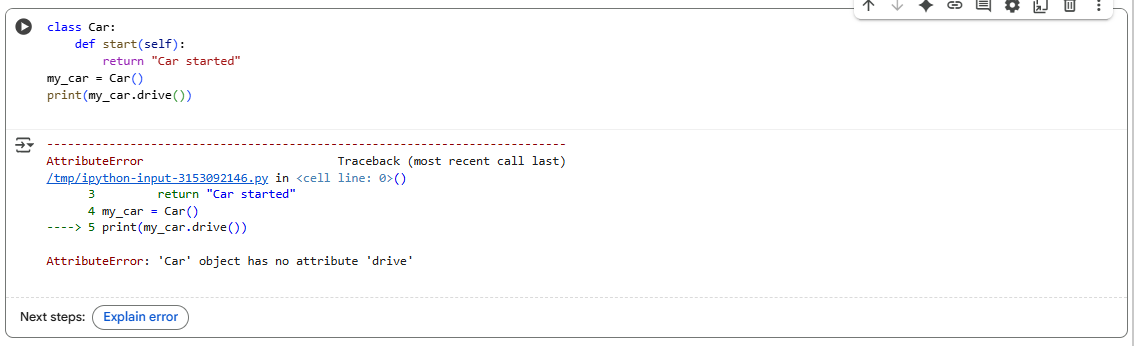
my\_car = Car()

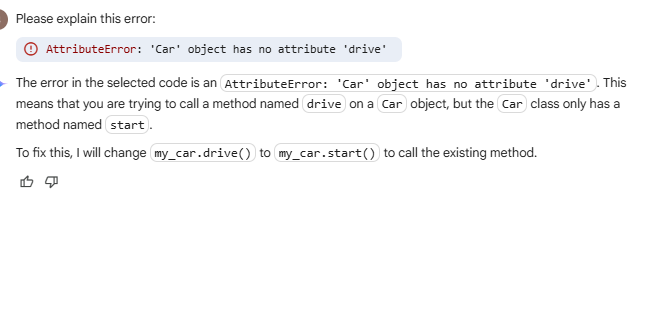
print(my\_car.drive()) #drive() is not defined

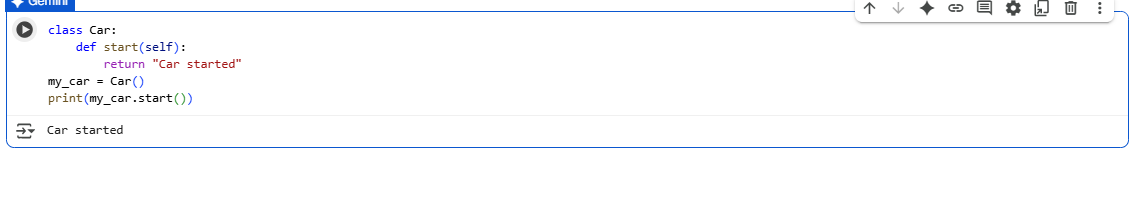
Requirements:

* Students must analyze whether to define the missing method or correct the method call.

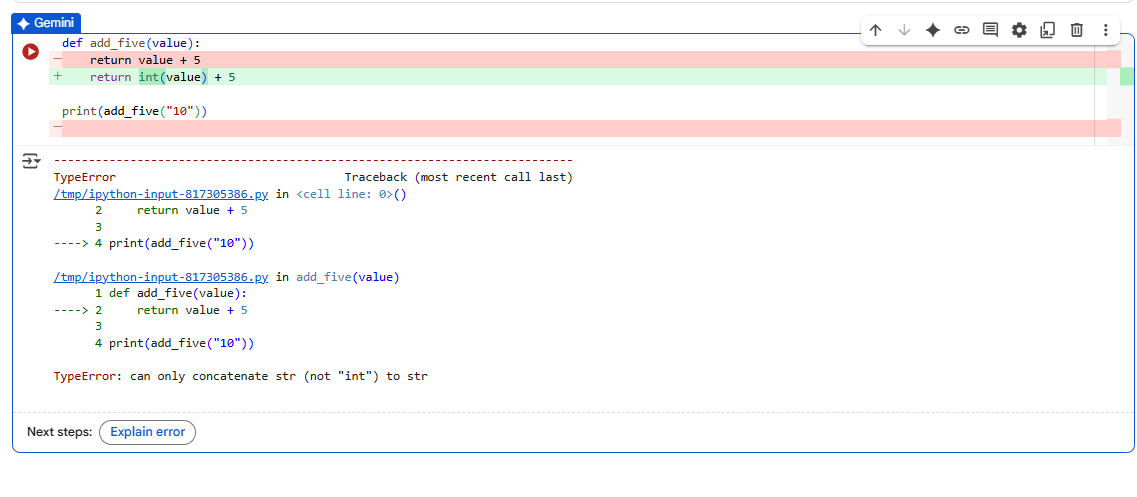
Use 3 assert tests to confirm the corrected class works

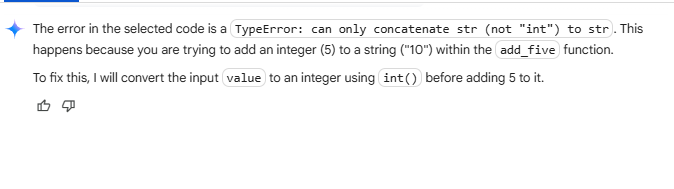


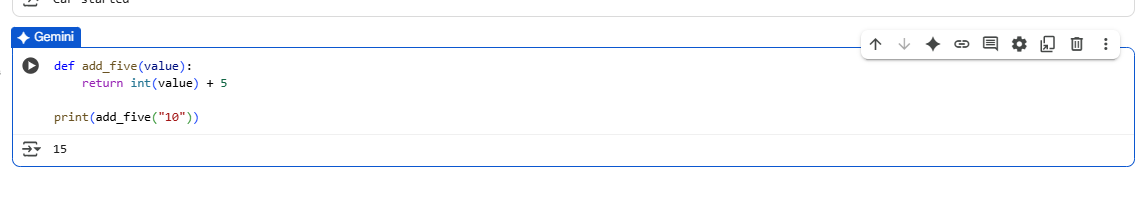




**Observation – Task 4:**  
In this task, I explored how an AttributeError occurs when calling a method that does not exist in a class. The buggy code attempted to call drive() on a Car object even though only start() was defined. With AI-assisted debugging, I understood two possible fixes: either define the missing method or correct the method call to an existing one. After correction and verification with assert test cases, the class worked as expected. This task emphasized the importance of object-oriented design clarity and showed how AI can simplify identifying undefined attributes or methods.







**Observation – Task 5:**  
In this task, I learned how a TypeError occurs when trying to add a string and an integer in Python. The AI tool explained why Python does not allow direct operations between incompatible data types. It also suggested two valid solutions: type casting the string into an integer for arithmetic addition, and converting the integer into a string for concatenation. By applying both methods and verifying with assert test cases, the code worked correctly. This task highlighted the importance of understanding data types and showed how AI can provide multiple approaches to fix type-related bugs.