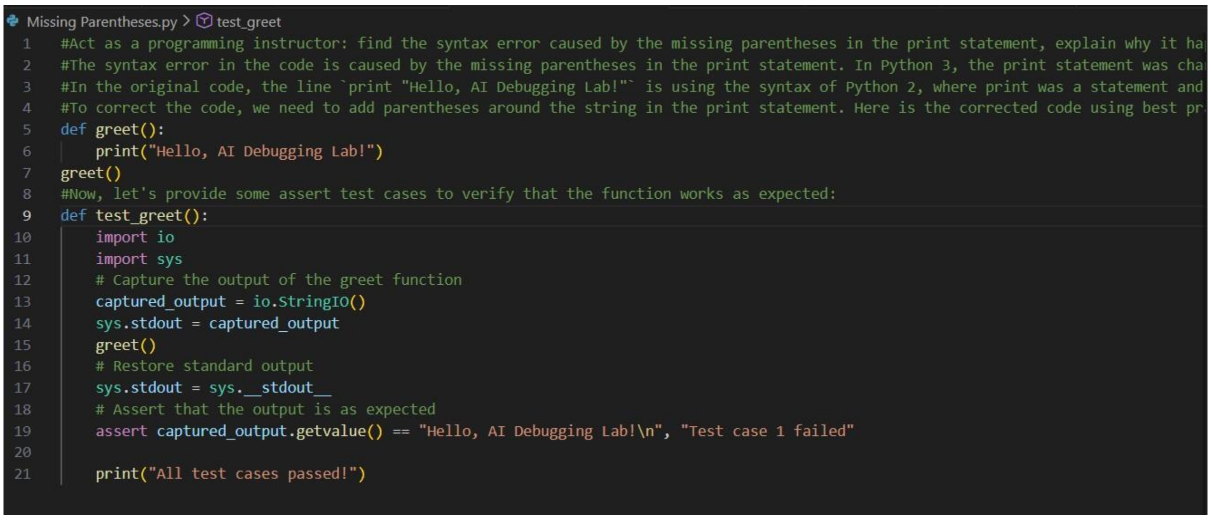
AI Assisted Coding -7

Name :Naga Chaitanya
HT No : 2303A52458
Batch No : 40
http://localhost:3000
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.
Expected Output #1:
• Corrected code with proper syntax and AI explanation.

Prompt :

```
#Act as a programming instructor: find the syntax error caused by the missing parentheses in the print statement, explain why it happens in Python 3, correct the code using best practices, and provide at least three assert test cases with expected output → def greet(): print "Hello, AI Debugging Lab!"; greet()
```

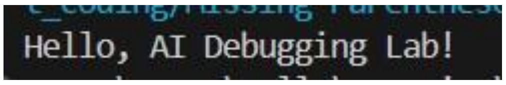
Code :



```
#Act as a programming instructor: find the syntax error caused by the missing parentheses in the print statement, explain why it ha
#The syntax error in the code is caused by the missing parentheses in the print statement. In Python 3, the print statement was cha
#In the original code, the line `print "Hello, AI Debugging Lab!"` is using the syntax of Python 2, where print was a statement and
#To correct the code, we need to add parentheses around the string in the print statement. Here is the corrected code using best pr
def greet():
    print("Hello, AI Debugging Lab!")
greet()
#Now, let's provide some assert test cases to verify that the function works as expected:
def test_greet():
    import io
    import sys
    # Capture the output of the greet function
    captured_output = io.StringIO()
    sys.stdout = captured_output
    greet()
    # Restore standard output
    sys.stdout = sys.__stdout__
    # Assert that the output is as expected
    assert captured_output.getvalue() == "Hello, AI Debugging Lab!\n", "Test case 1 failed"

    print("All test cases passed!")
```

OUTPUT :



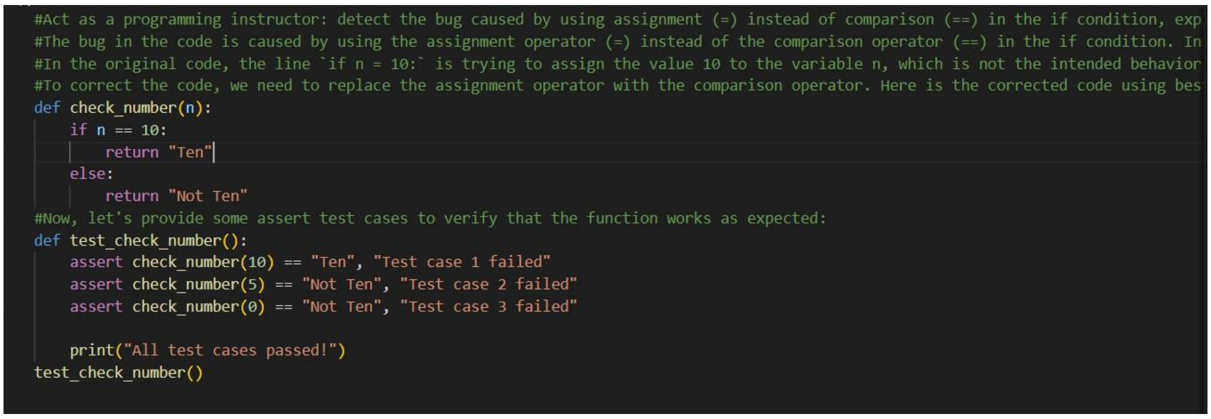
```
Hello, AI Debugging Lab!
```

Task Description #2 (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 :**

#Act as a programming instructor: detect the bug caused by using assignment (=) instead of comparison (==) in the if condition, explain why it is wrong, correct the code using best practices, and provide at least three assert test cases with expected output → def check_number(n): if n = 10: return "Ten" else: return "Not Ten"

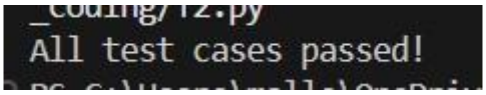
Code :



```python
#Act as a programming instructor: detect the bug caused by using assignment (=) instead of comparison (==) in the if condition, exp
#The bug in the code is caused by using the assignment operator (=) instead of the comparison operator (==) in the if condition. In
#In the original code, the line `if n = 10:` is trying to assign the value 10 to the variable n, which is not the intended behavior
#To correct the code, we need to replace the assignment operator with the comparison operator. Here is the corrected code using bes
def check_number(n):
    if n == 10:
        return "Ten"
    else:
        return "Not Ten"
#Now, let's provide some assert test cases to verify that the function works as expected:
def test_check_number():
    assert check_number(10) == "Ten", "Test case 1 failed"
    assert check_number(5) == "Not Ten", "Test case 2 failed"
    assert check_number(0) == "Not Ten", "Test case 3 failed"

    print("All test cases passed!")
test_check_number()
```

OUTPUT :



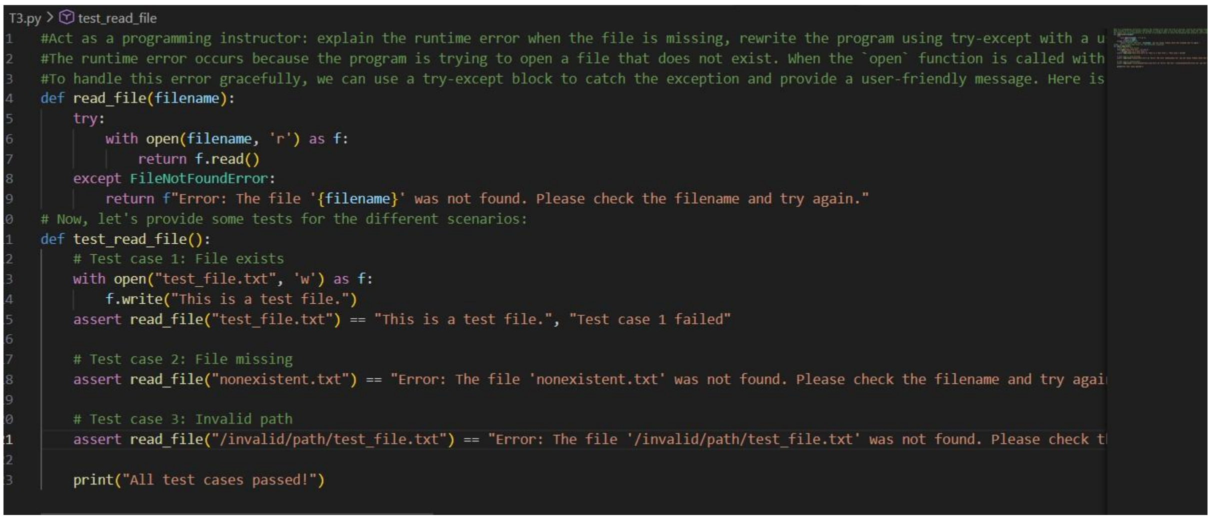
```
_coding/T2.py
All test cases passed!
```

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.
Expected Output #3:
• Safe file handling with exception management.

Prompt :

#Act as a programming instructor: explain the runtime error when the file is missing, rewrite the program using try-except with a userfriendly message, and provide tests for file exists, file missing, and invalid path along with expected output → def read_file(filename): with open(filename,'r') as f: return f.read(); print(read_file("nonexistent.txt"))

Code :



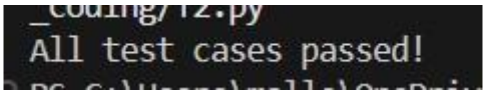
```
T3.py > test_read_file
#Act as a programming instructor: explain the runtime error when the file is missing, rewrite the program using try-except with a u
#The runtime error occurs because the program is trying to open a file that does not exist. When the `open` function is called with
#To handle this error gracefully, we can use a try-except block to catch the exception and provide a user-friendly message. Here is
def read_file(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found. Please check the filename and try again."
# Now, let's provide some tests for the different scenarios:
def test_read_file():
    # Test case 1: File exists
    with open("test_file.txt", 'w') as f:
        f.write("This is a test file.")
    assert read_file("test_file.txt") == "This is a test file.", "Test case 1 failed"

    # Test case 2: File missing
    assert read_file("nonexistent.txt") == "Error: The file 'nonexistent.txt' was not found. Please check the filename and try agai

    # Test case 3: Invalid path
    assert read_file("/invalid/path/test_file.txt") == "Error: The file '/invalid/path/test_file.txt' was not found. Please check t

    print("All test cases passed!")
```

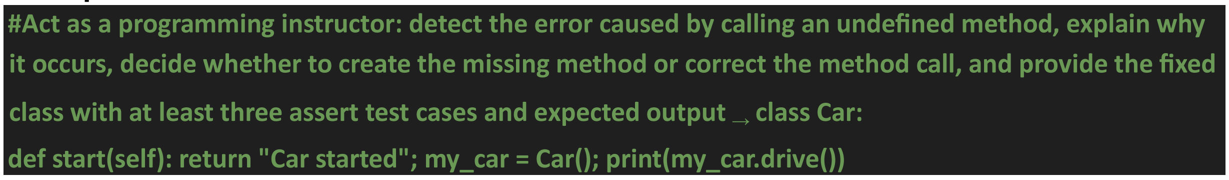
OUTPUT :



```
_coding/T2.py
All test cases passed!
```

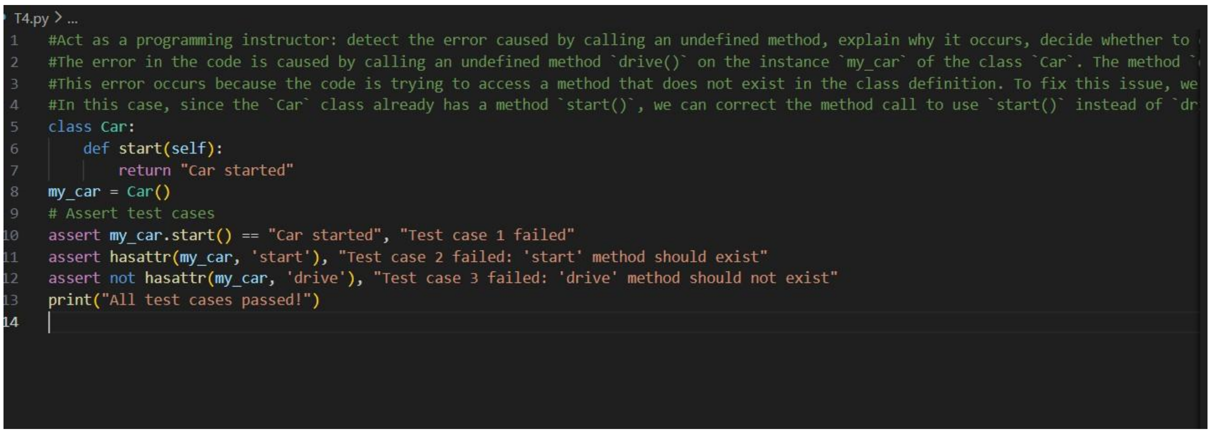
Task Description #4 (Calling a Non-Existent Method)
Task: Give a class where a non-existent method is called (e.g., obj.undefined_method()). Use AI to debug and fix.
# Bug: Calling an undefined method class
Car:
def start(self): return
"Car started"
my_car = Car()
print(my_car.drive()) # drive() is not 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 :



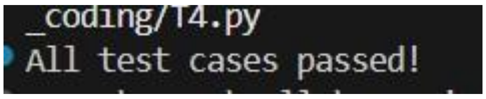
```
#Act as a programming instructor: detect the error caused by calling an undefined method, explain why it occurs, decide whether to create the missing method or correct the method call, and provide the fixed class with at least three assert test cases and expected output → class Car:
def start(self): return "Car started"; my_car = Car(); print(my_car.drive())
```

Code :



```
T4.py > ...
#Act as a programming instructor: detect the error caused by calling an undefined method, explain why it occurs, decide whether to
#The error in the code is caused by calling an undefined method `drive()` on the instance `my_car` of the class `Car`. The method `
#This error occurs because the code is trying to access a method that does not exist in the class definition. To fix this issue, we
#In this case, since the `Car` class already has a method `start()`, we can correct the method call to use `start()` instead of `dr
class Car:
    def start(self):
        return "Car started"
my_car = Car()
# Assert test cases
assert my_car.start() == "Car started", "Test case 1 failed"
assert hasattr(my_car, 'start'), "Test case 2 failed: 'start' method should exist"
assert not hasattr(my_car, 'drive'), "Test case 3 failed: 'drive' method should not exist"
print("All test cases passed!")
```

OUTPUT :



```
_coding/T4.py
All test cases passed!
```

Task Description #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.

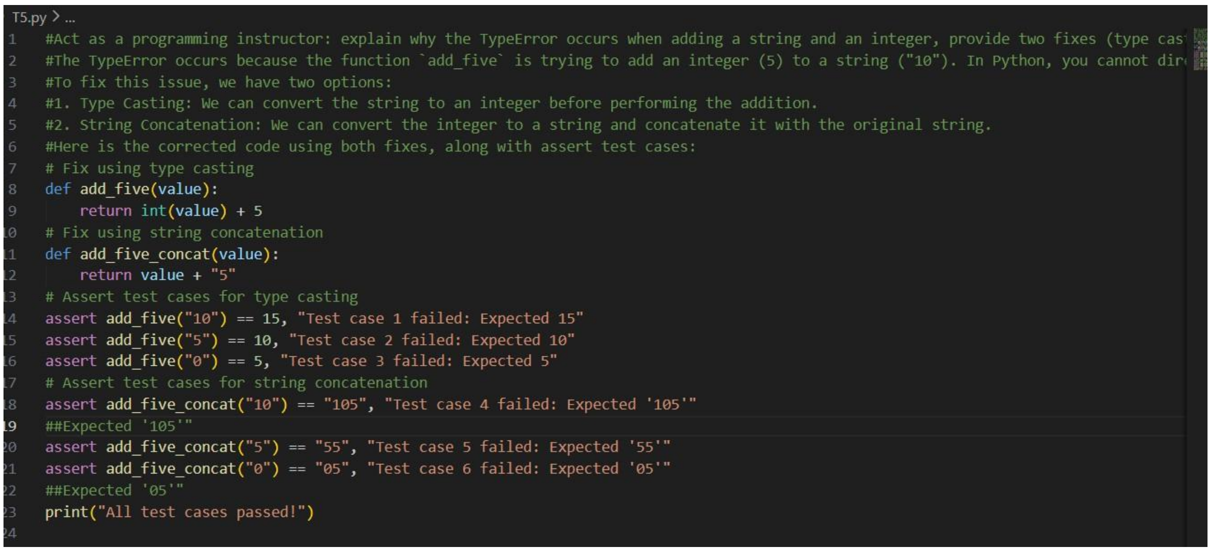
Expected Output #5:

- Corrected code that runs successfully for multiple inputs.

Prompt :

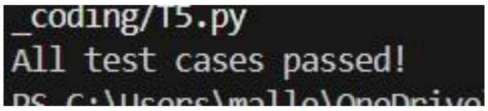
#Act as a programming instructor: explain why the TypeError occurs when adding a string and an integer, provide two fixes (type casting and string concatenation), and give corrected code with at least three assert test cases and expected output → def add_five(value): return value + 5; print(add_five("10"))

Code :



```
T5.py > ...
#Act as a programming instructor: explain why the TypeError occurs when adding a string and an integer, provide two fixes (type cas
#The TypeError occurs because the function `add_five` is trying to add an integer (5) to a string ("10"). In Python, you cannot dir
#To fix this issue, we have two options:
#1. Type Casting: We can convert the string to an integer before performing the addition.
#2. String Concatenation: We can convert the integer to a string and concatenate it with the original string.
#Here is the corrected code using both fixes, along with assert test cases:
# Fix using type casting
def add_five(value):
    return int(value) + 5
# Fix using string concatenation
def add_five_concat(value):
    return value + "5"
# Assert test cases for type casting
assert add_five("10") == 15, "Test case 1 failed: Expected 15"
assert add_five("5") == 10, "Test case 2 failed: Expected 10"
assert add_five("0") == 5, "Test case 3 failed: Expected 5"
# Assert test cases for string concatenation
assert add_five_concat("10") == "105", "Test case 4 failed: Expected '105'"
##Expected '105'"
assert add_five_concat("5") == "55", "Test case 5 failed: Expected '55'"
assert add_five_concat("0") == "05", "Test case 6 failed: Expected '05'"
##Expected '05'"
print("All test cases passed!")
```

OUTPUT :



```
_coding/T5.py
All test cases passed!
```