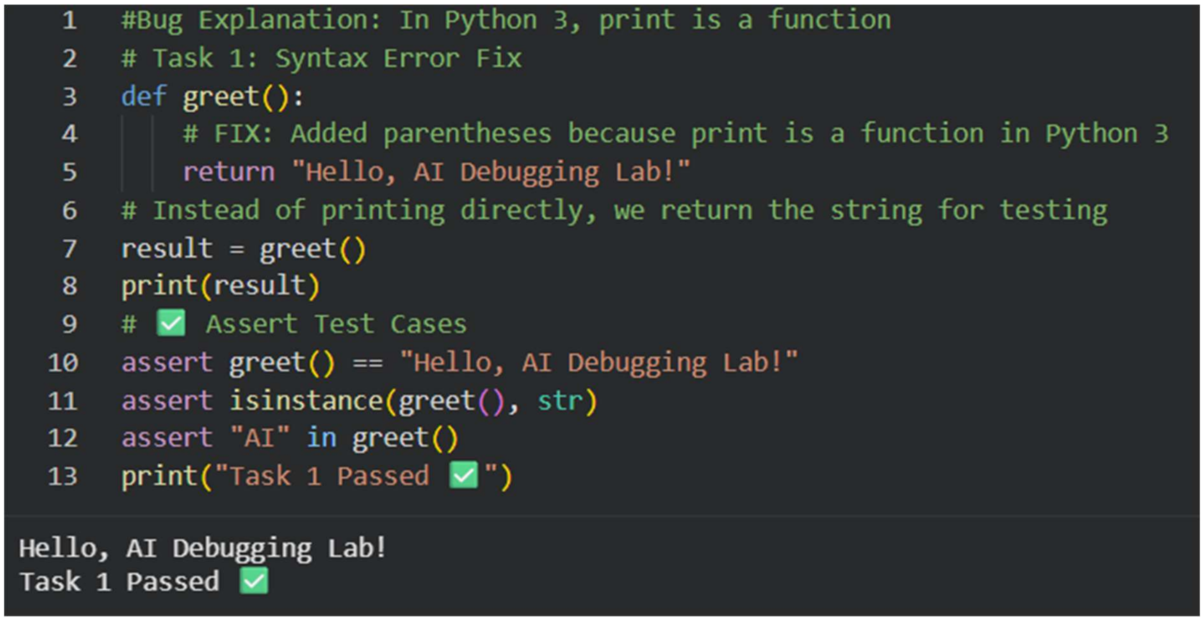
ASSIGNMENT-7.1

Name: J.Vyshnavi

HT. No: 2303A51895

Batch: 08
Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs
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.

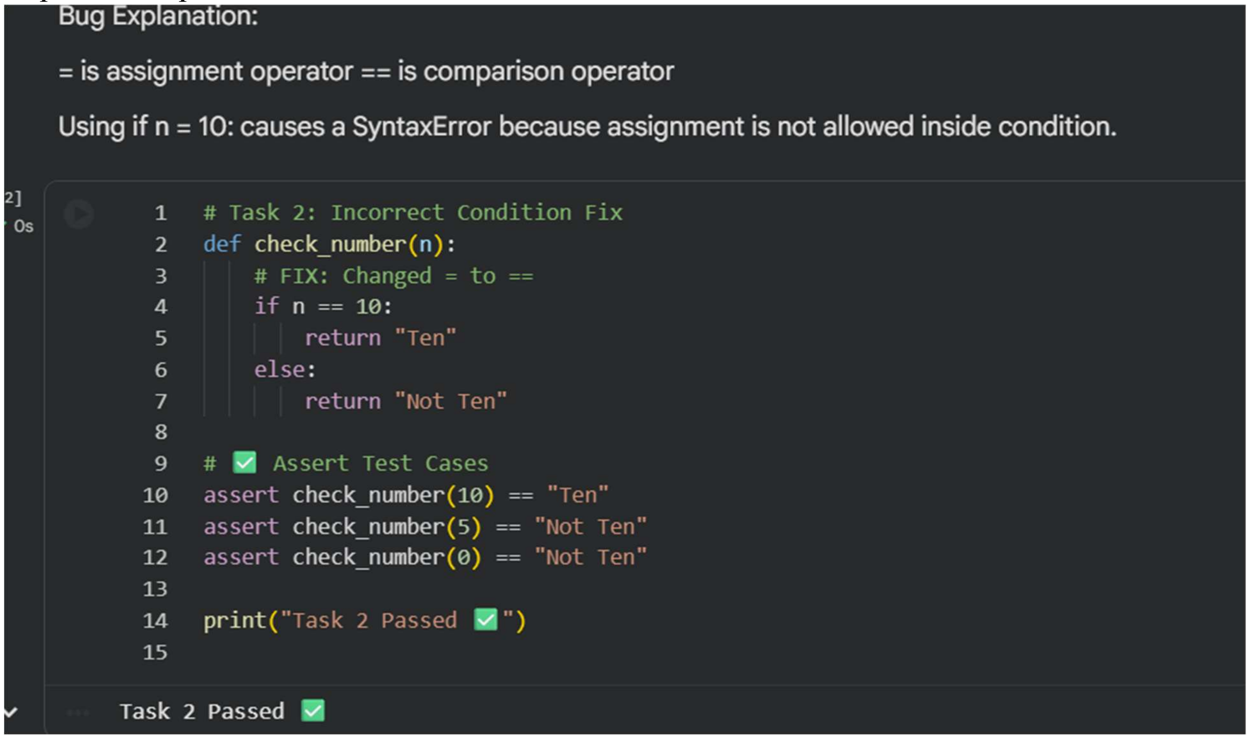


```python
#Bug Explanation: In Python 3, print is a function
# Task 1: Syntax Error Fix
def greet():
    # FIX: Added parentheses because print is a function in Python 3
    return "Hello, AI Debugging Lab!"
# Instead of printing directly, we return the string for testing
result = greet()
print(result)
# ✅ Assert Test Cases
assert greet() == "Hello, AI Debugging Lab!"
assert isinstance(greet(), str)
assert "AI" in greet()
print("Task 1 Passed ✅")
```

```
Hello, AI Debugging Lab!
Task 1 Passed ✅
```

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 :



Bug Explanation:

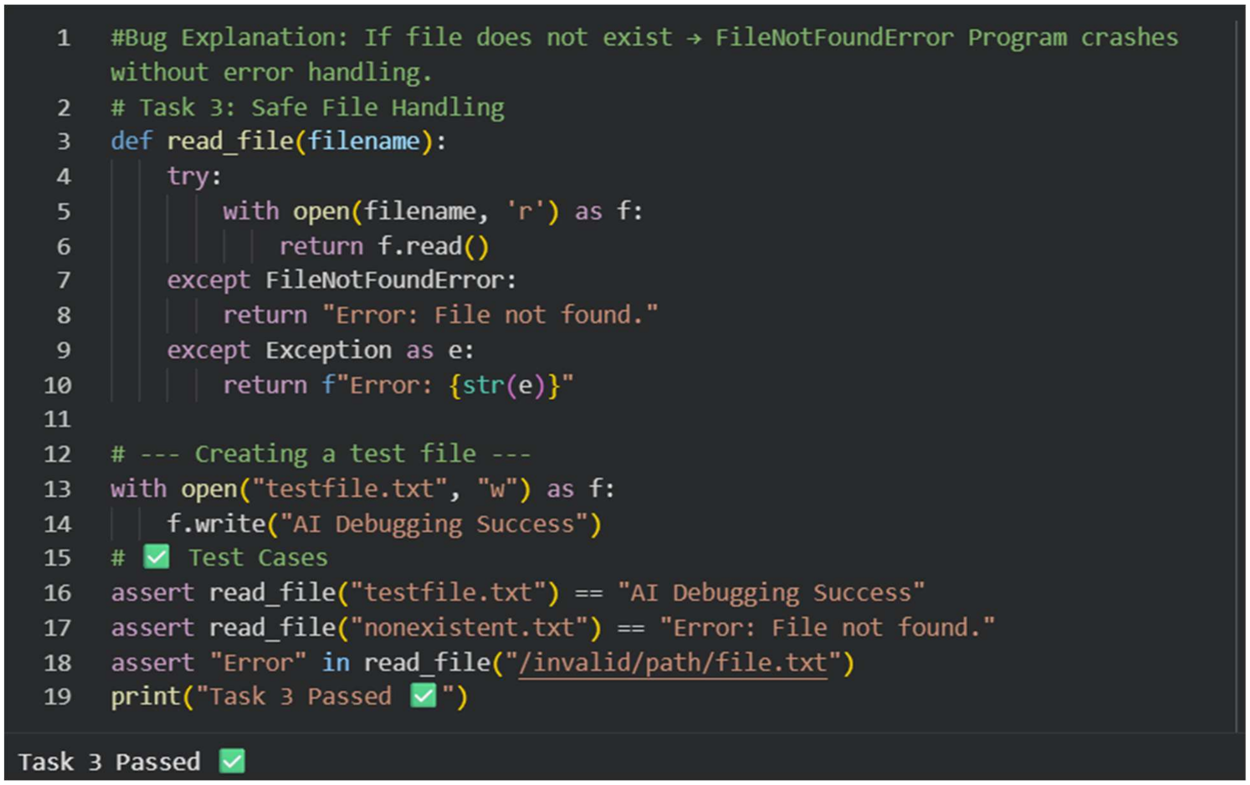
= is assignment operator == is comparison operator

Using if n = 10: causes a SyntaxError because assignment is not allowed inside condition.

```
# Task 2: Incorrect Condition Fix
def check_number(n):
    # FIX: Changed = to ==
    if n == 10:
        return "Ten"
    else:
        return "Not Ten"

# ✅ Assert Test Cases
assert check_number(10) == "Ten"
assert check_number(5) == "Not Ten"
assert check_number(0) == "Not Ten"

print("Task 2 Passed ✅")

```

```
Task 2 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:



```python
#Bug Explanation: If file does not exist → FileNotFoundError Program crashes without error handling.
# Task 3: Safe File Handling
def read_file(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return "Error: File not found."
    except Exception as e:
        return f"Error: {str(e)}"

# --- Creating a test file ---
with open("testfile.txt", "w") as f:
    f.write("AI Debugging Success")
# ✅ Test Cases
assert read_file("testfile.txt") == "AI Debugging Success"
assert read_file("nonexistent.txt") == "Error: File not found."
assert "Error" in read_file("/invalid/path/file.txt")
print("Task 3 Passed ✅")
```

```
Task 3 Passed ✅
```

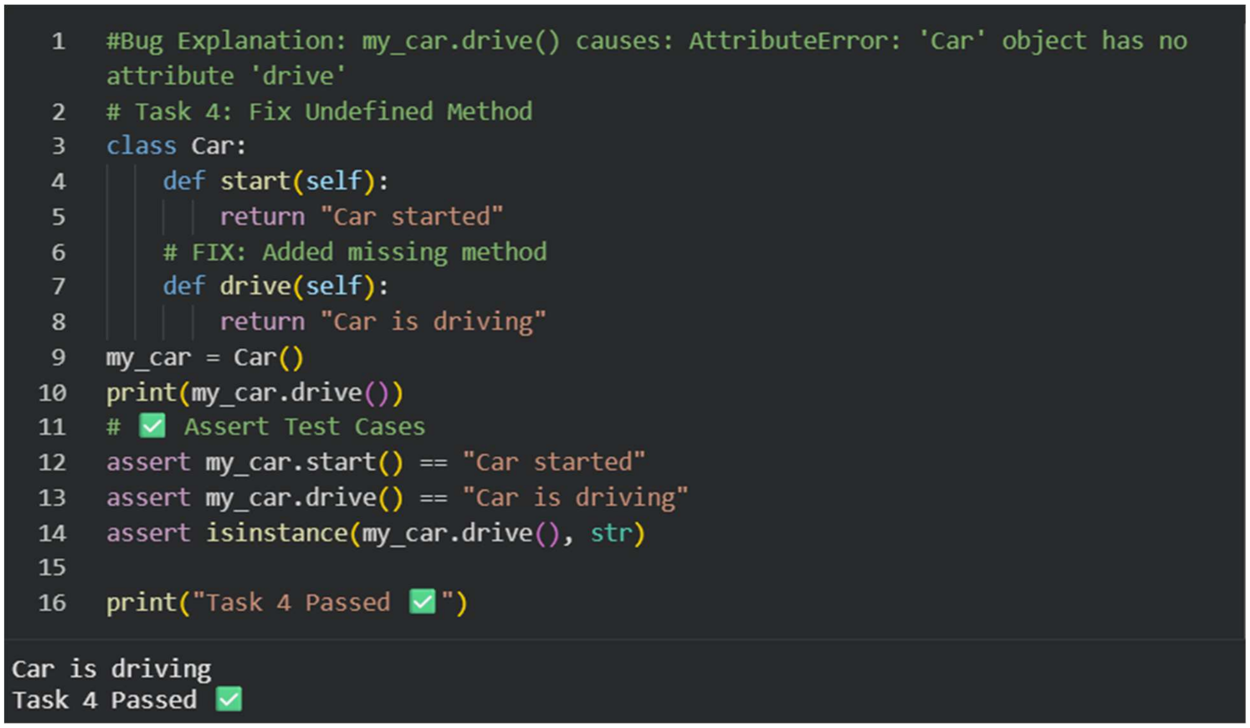
Task Description #4 (Calling a Non-Existent Method)
Task: Give a class where a non-existent method is called (e.g., obj.undefined_method()). Use AI to debug and fix.

# 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.

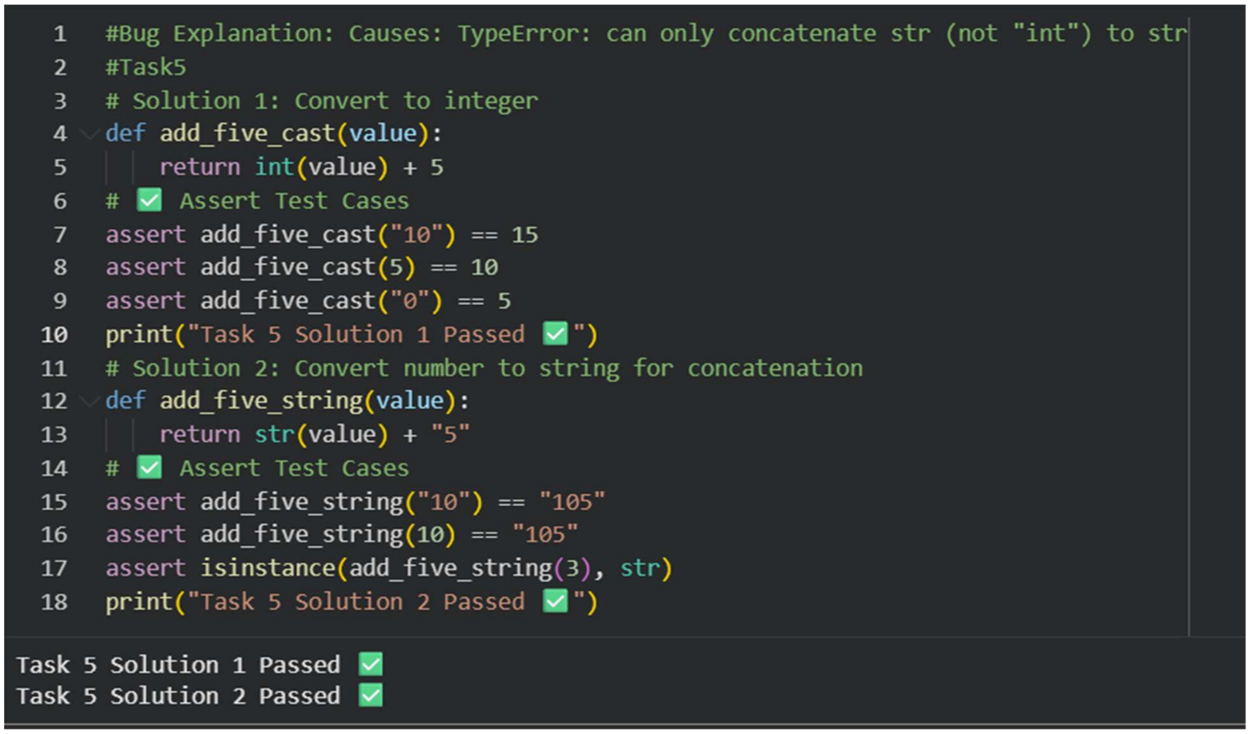
Expected Output:



```python
#Bug Explanation: my_car.drive() causes: AttributeError: 'Car' object has no attribute 'drive'
# Task 4: Fix Undefined Method
class Car:
    def start(self):
        return "Car started"
    # FIX: Added missing method
    def drive(self):
        return "Car is driving"
my_car = Car()
print(my_car.drive())
# ✅ Assert Test Cases
assert my_car.start() == "Car started"
assert my_car.drive() == "Car is driving"
assert isinstance(my_car.drive(), str)

print("Task 4 Passed ✅")
```

```
Car is driving
Task 4 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:



```python
#Bug Explanation: Causes: TypeError: can only concatenate str (not "int") to str
#Task5
# Solution 1: Convert to integer
def add_five_cast(value):
    return int(value) + 5
# ✅ Assert Test Cases
assert add_five_cast("10") == 15
assert add_five_cast(5) == 10
assert add_five_cast("0") == 5
print("Task 5 Solution 1 Passed ✅")
# Solution 2: Convert number to string for concatenation
def add_five_string(value):
    return str(value) + "5"
# ✅ Assert Test Cases
assert add_five_string("10") == "105"
assert add_five_string(10) == "105"
assert isinstance(add_five_string(3), str)
print("Task 5 Solution 2 Passed ✅")
```

```
Task 5 Solution 1 Passed ✅
Task 5 Solution 2 Passed ✅
```