ASSIGNMENT-7.1

M.PARAMESH

HNO: 2303A53021

BATCH-46

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 Al to detect and fix the syntax error.

Bug: Missing parentheses in print statement def greet():

print "Hello, Al Debugging Lab!"

greet()

Requirements:

- Run the given code to observe the error.
- Apply Al 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 Al explanation.

PROMPT:

#Detect and fix the syntax error in this Python code:

#def greet(): print 'Hello, Al Debugging Lab!' ; greet() #Explain the issue, provide corrected Python 3 code, and include 3 assert tests to verify correct output.

CODE:

```
def test_greet():
  captured_output = io.StringIO()        #  Create  StringIO  object
  sys.stdout = captured_output # Redirect stdout.
 greet()                  # Call the function.
   sys.stdout = sys._stdout_        # Reset redirect.
assert captured_output.getvalue().strip() == 'Hello, Al Debugging Lab!'
```

test_greet()

def test_greet_output():

captured_output = io.StringIO() sys.stdout = captured_output

greet() sys.stdout = sys._stdout_ assert

captured_output.getvalue() 'Hello, Al DebuggingLab!\n'

test_greet_output() def

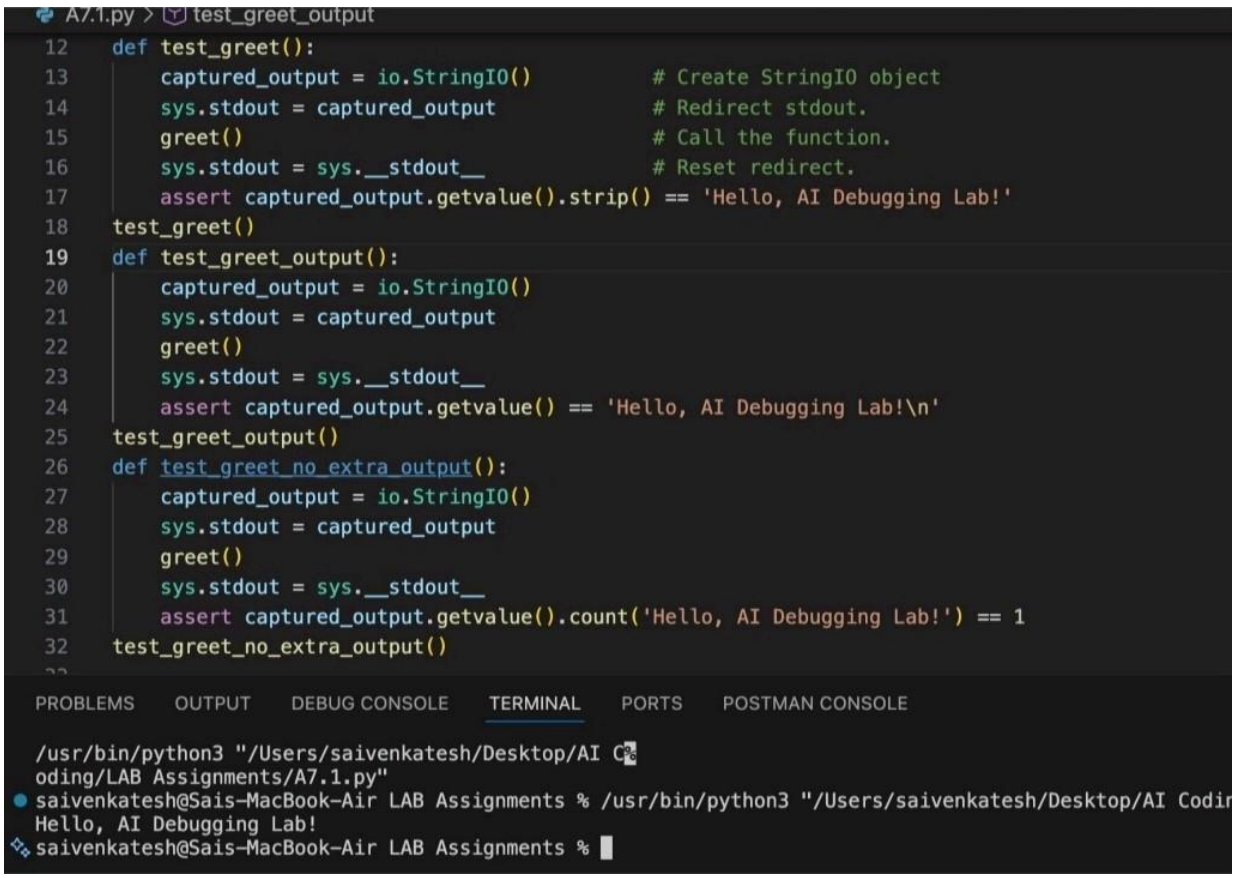
test_greet_no_extra_output():

captured_output = io.StringIO() sys.stdout = captured_output greet()

sys.stdout = sys._stdout_ assert

captured_output.getvalue().count('Hello, Al Debugging Lab") == 1

test_greet_no_extra_output() output:



Explanation:

The issue in the original code is that it uses Python 2 syntax for the print statement. In Python 3, print is a function and requires parentheses.

Task Description #2 (Incorrect condition in an If Statement)

Task: Supply a function where an if-condition mistakenly uses = instead of Let Al 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 Al 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.

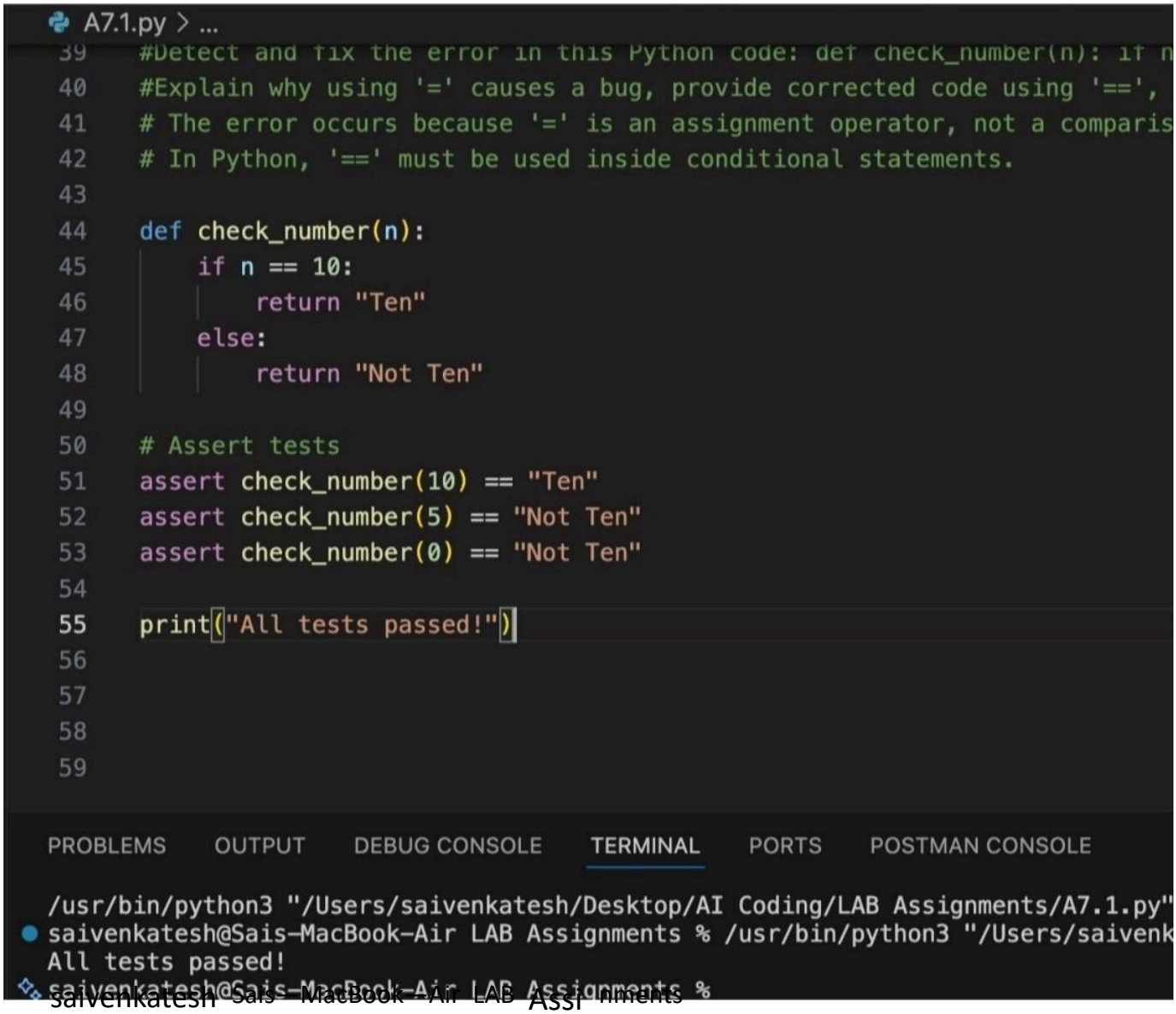
Prompt used:

Detect and fix the error in this Python code: def check_number(n): if n = 10: return "Ten" else: return "Not Ten". #Explain why using causes a bug, provide corrected code using '==', and include 3 assert tests to verify the function works. CODE:

def check_number(n): if n == 10. return "Ten" else: return "Not Ten"

Assert tests

assert check_number(10) == "Ten" assert check_number(5) == "Not Ten" assert check_number(O) == "Not Ten" print("All tests passed! ") output:



Explaination :

The error occurs because = is an assignment operator, not a comparison operator. In an if condition, Python requires == to compare values, and using = causes a SyntaxError. Replacing = with == correctly checks whether n is equal to 10, allowing the function to work properly.

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:

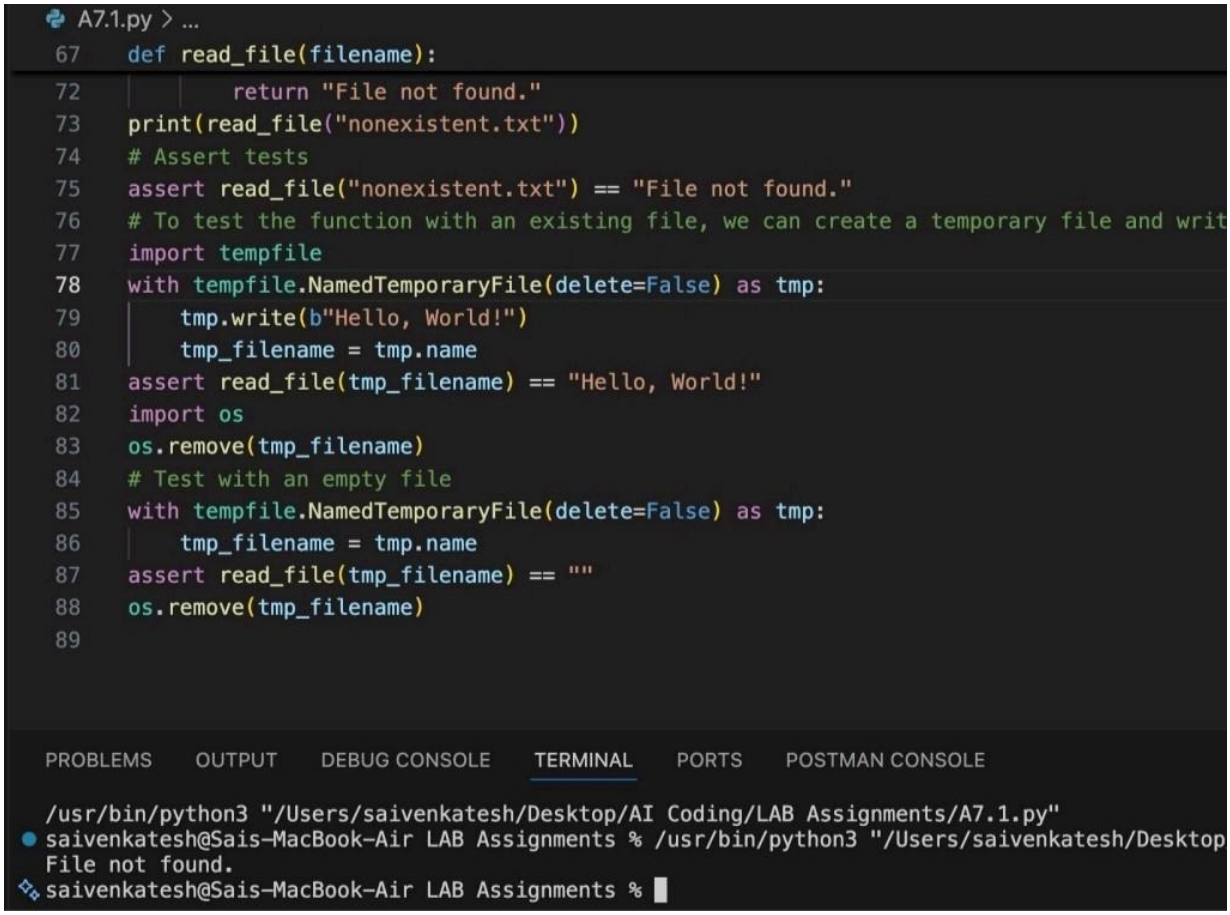
#Analyze this Python code that crashes when a file does not exist: def read_file(filename): with open(filename,'r') as f: return f.read()

#Fix the runtime error using try—except, add a user-friendly error message, and include 3 assert tests for file exists, file missing, and invalid path cases.

code:

```
def read_file(filename):
  try:
    with open(filename, "r") as file:
      return file. read()
  except FileNotFoundError:
    return "File not found."


print(read_file("nonexistent.txt"))


# Assert tests assert read_file("nonexistent.txt") ==
"File not found."


# To test the function with an existing file, we can create a temporary file and write to it
import tempfile


with tempfile.NamedTemporaryFile(delete=False) as tmp:
  tmp.write(b"Hello, World!")
  tmp_filename = tmp.name
```

```
assert read_file(tmp_filename) "Hello, World!"


import os
os.remove(tmp_filename)


# Test with an empty file with
tempfile.NamedTemporaryFile(delete=False) as tmp:
   tmp_filename = tmp.name


assert read_file(tmp_filename) " "
os.remove(tmp_filename) output :
```



Explaination:

The program crashes because it tries to open a file that does not exist, which raises a FileNotFoundError. By using a try—except block, the exception is caught and handled safely instead of stopping the program. This allows the function to return a userfriendly error message while continuing execution.

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 Al 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 used:

#Detect and fix the error in this Python code where a nonexistent method is called: class Car: def start(self): return "Car started" my_car = Car(); print(my_car.drive()) #Explain the cause of the error, decide whether to define the missing method or correct the method call, provide corrected code, and include 3 assert tests to verify the class works properly.

Code :

```
class Car:
  def sta rt(self):
    return "Car started"

  def drive(self):
    return "Car is driving"



my_car = Car()
print(my_car.drive())


# Assert tests assert my_car.start() ==
"Car started" assert my_car.drive() ==
"Car is driving"


another_car = Car()
assert another_car.start() "Car started" assert
another_car.drive() "Car is driving" output:
```

explaination:

The error occurs because the method drive() is called on the Car object, but it is not defined in the Car class, which raises an

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 Al 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 Al 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 used:

#Detect and fix the TypeError in this Python code caused by adding a string and an integer: def add five(value): return value + 5 ; print(add_five("10"))#Explain why the error occurs, provide two solutions (one using type casting for numeric addition and one using string concatenation), and include 3 assert tests to verify both solutions work correctly.

Code:

Detect and fix the TypeError in this Python code caused by adding a string and an integer:

def add_five(value): return value + 5 ; print(add_five(" 10"))

The error occurs because the code is trying to add an integer (5)

to a string (" 10"), which is not allowed in Python.

Solution l : Using type casting for numeric addition

def dd_five_numeric(value):

    return int(value) + 5

```
print(add_five_numeric(" I O" ))


# Assert tests for numeric addition assert
add_five_numeric(" I O") 15
        ıdd_five_numeric("0") ==
        ıdd_five_numeric("-5") ==
assert 5 assert O


# Solution 2: Using string concatenation def
add_five_string(value):
    return value + "5"

print(add_five_string("10"))


# Assert tests for string concatenation
assert      _  add five_string(" 10") " 105"
assert add_ five_string("Hello") "He1105"
assert      _  add five_string("") — "5"
output:
```

Explaination:
The error occurs because Python cannot add a string and an integer, which raises a TypeError. One solution is to convert the input to an integer

using type casting so numeric addition can be performed. Another solution is to convert the value to a string and use string concatenation instead, depending on the intended behavior.

```
#def  return value + 5 ; print
#Explain why the error occurs, provide two solutions (one using type casting for numeric additi0[
# The error occurs because the code is trying to add an integer (5) to a string       which is 118  #
Solution 1: Using type casting for numeric addition
      def       :
      return int(value) + 5
```

```
print ( add_f ive_numeric 10" ) )
# Assert tests for numeric addition
assert add_five_numeric( " 10 " ) --— 15
assert add_five_numeric(        )
assert add_five_numeric("-5")
# Solution 2: Using string concatenation 127        def add_five_string(vatue) :
return value + "5"
print ( add_f ive_string (    )
# Assert tests for string concatenation
assert   "105
assert add_f ive_st ring ("Het )       ''He1105"
assert add_five_string( 134
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assignments/A7.1.py"
• saivenkatesh@Sais—MacBook—Air LAB Assignments % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/L 15
105
```