

Name - Gutha Srinidhi
H.No - 2303A53023 - B46

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

Lab Objectives:

- To identify and correct syntax, logic, and runtime errors in Python programs using AI tools.
- To understand common programming bugs and AI-assisted debugging suggestions.
- To evaluate how AI explains, detects, and fixes different types of coding errors.
- To build confidence in using AI to perform structured debugging practices.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Use AI tools to detect and correct syntax, logic, and runtime errors.
- Interpret AI-suggested bug fixes and explanations.
- Apply systematic debugging strategies supported by AI-generated insights.
- Refactor buggy code using responsible and reliable programming patterns.

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:

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

#def greet(): print 'Hello, AI Debugging Lab!' ; greet()

#Explain the issue, provide corrected Python 3 code, and include 3 assert tests to verify correct output.

CODE:

```
def greet():
```

```
    print('Hello, AI Debugging Lab!')
```

```
greet()
```

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.

Assert tests to verify correct output

```
import io
```

```
import sys
```

```
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, AI 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, AI Debugging Lab!
\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, AI Debugging
Lab!') == 1
test_greet_no_extra_output()

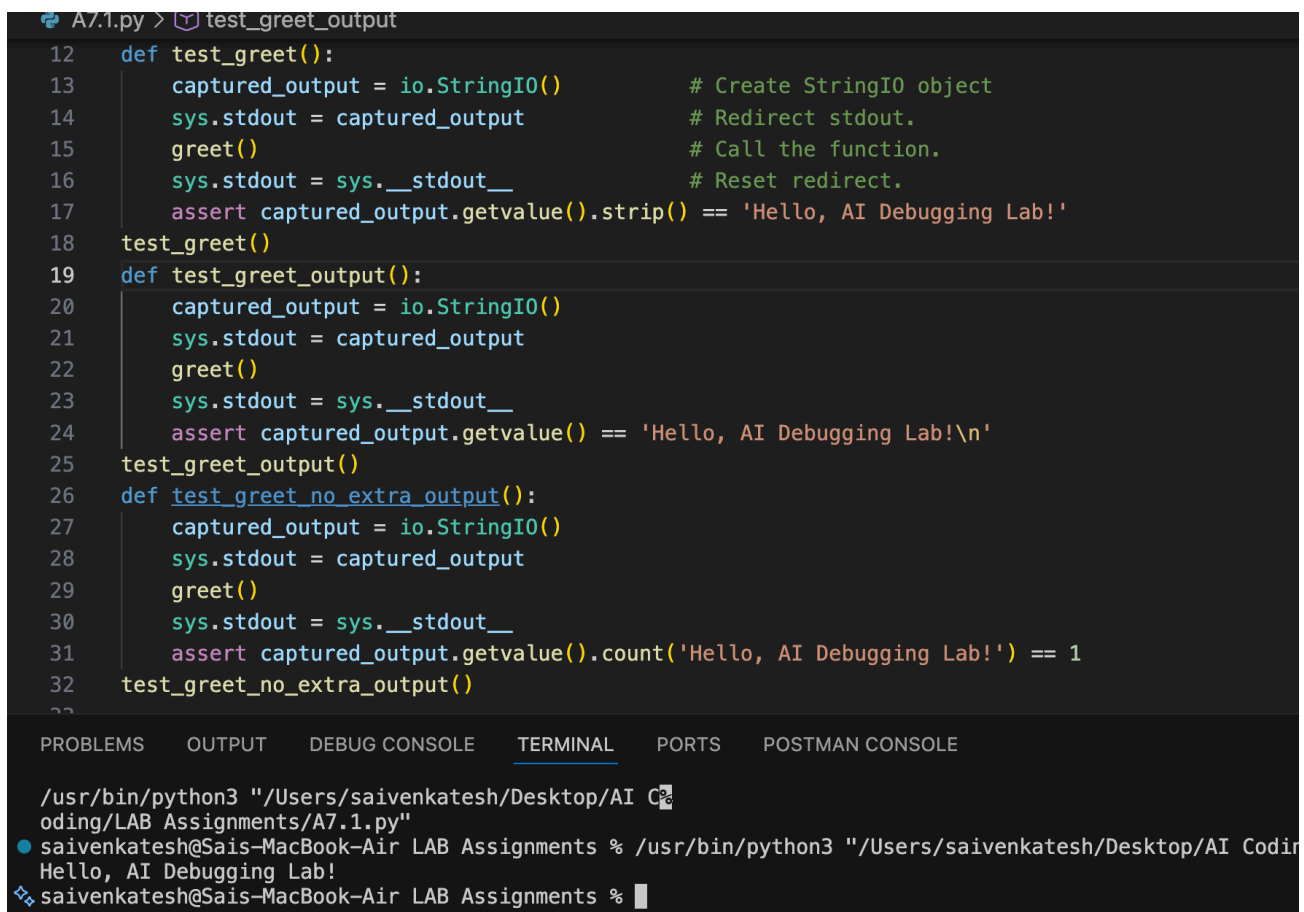
```

OUTPUT:

Hello, AI Debugging Lab!

AI 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.



The screenshot shows a code editor with a dark theme. The top part displays Python code for testing a 'greet' function. The code defines 'test_greet()' and 'test_greet_output()' which capture stdout using 'io.StringIO()'. It then defines 'test_greet_no_extra_output()' which checks the count of 'Hello, AI Debugging Lab!' in the captured output. The bottom part shows the terminal output of running the code, which prints 'Hello, AI Debugging Lab!'.

```

A7.1.py > test_greet_output
12 def test_greet():
13     captured_output = io.StringIO()           # Create StringIO object
14     sys.stdout = captured_output              # Redirect stdout.
15     greet()                                  # Call the function.
16     sys.stdout = sys.__stdout__              # Reset redirect.
17     assert captured_output.getvalue().strip() == 'Hello, AI Debugging Lab!'
18 test_greet()
19 def test_greet_output():
20     captured_output = io.StringIO()
21     sys.stdout = captured_output
22     greet()
23     sys.stdout = sys.__stdout__
24     assert captured_output.getvalue() == 'Hello, AI Debugging Lab!\n'
25 test_greet_output()
26 def test_greet_no_extra_output():
27     captured_output = io.StringIO()
28     sys.stdout = captured_output
29     greet()
30     sys.stdout = sys.__stdout__
31     assert captured_output.getvalue().count('Hello, AI Debugging Lab!') == 1
32 test_greet_no_extra_output()
33
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/LAB Assignments/A7.1.py"
Hello, AI Debugging Lab!
saivenkatesh@Sais-MacBook-Air LAB Assignments %

```

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:

#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(0) == "Not Ten"
```

```
print("All tests passed!")
```

OUTPUT:

All tests passed!

EXPLANATION

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.

A7.1.py > ...

```
39 #Detect and fix the error in this Python code: def check_number(n): if n = 10: re
40 #Explain why using '=' causes a bug, provide corrected code using '==', and inclu
41 # The error occurs because '=' is an assignment operator, not a comparison operat
42 # In Python, '==' must be used inside conditional statements.
43
44 def check_number(n):
45     if n == 10:
46         return "Ten"
47     else:
48         return "Not Ten"
49
50 # Assert tests
51 assert check_number(10) == "Ten"
52 assert check_number(5) == "Not Ten"
53 assert check_number(0) == "Not Ten"
54
55 print("All tests passed!")
56
57
58
59
```

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/Des
All tests passed!

saivenkatesh@Sais-MacBook-Air LAB Assignments %

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 f:  
            return f.read()
```

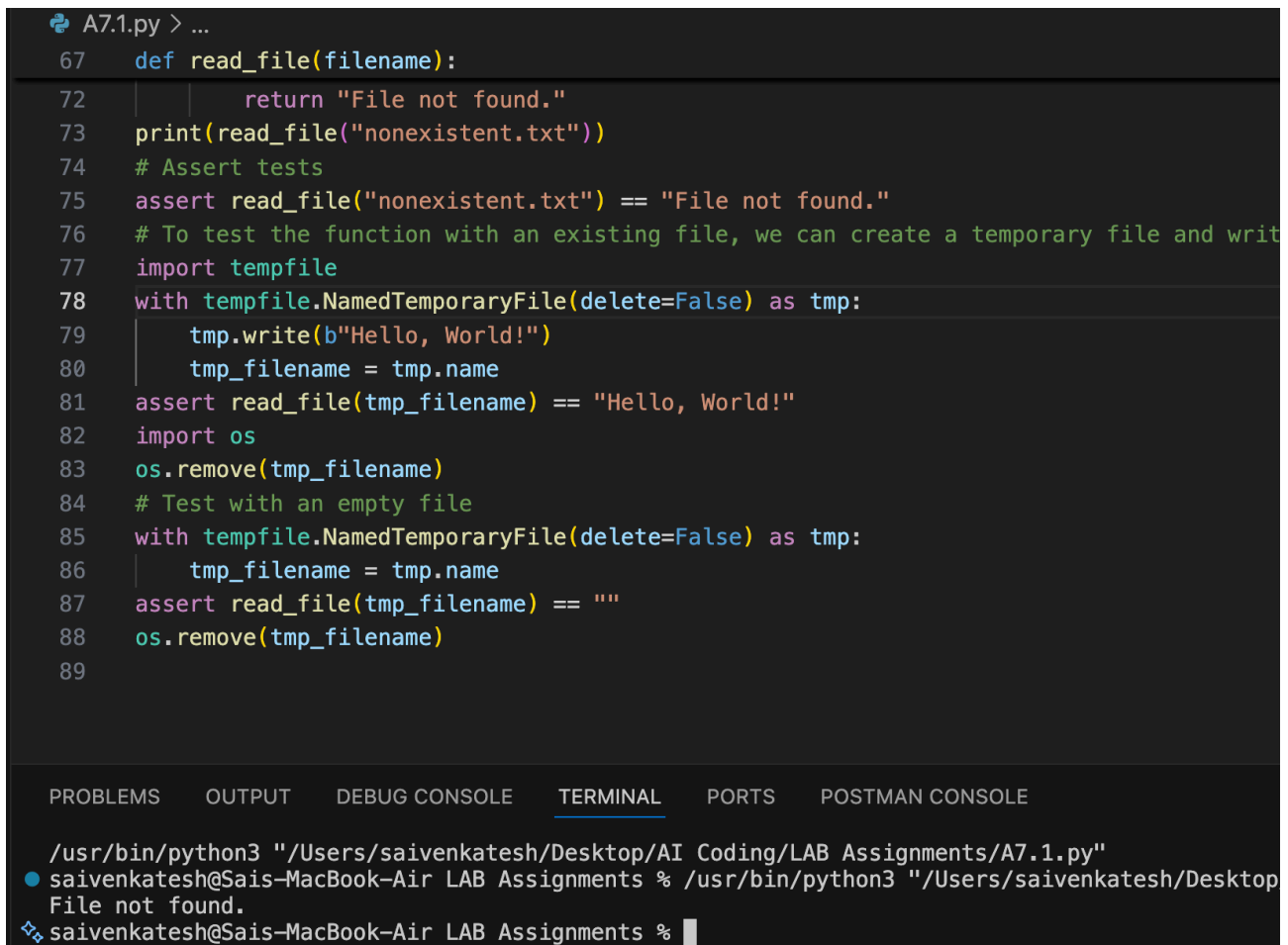
```

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 some content 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

File not found.



```

A7.1.py > ...
67  def read_file(filename):
72      |         return "File not found."
73  print(read_file("nonexistent.txt"))
74  # Assert tests
75  assert read_file("nonexistent.txt") == "File not found."
76  # To test the function with an existing file, we can create a temporary file and writ
77  import tempfile
78  with tempfile.NamedTemporaryFile(delete=False) as tmp:
79      |     tmp.write(b"Hello, World!")
80      |     tmp_filename = tmp.name
81  assert read_file(tmp_filename) == "Hello, World!"
82  import os
83  os.remove(tmp_filename)
84  # Test with an empty file
85  with tempfile.NamedTemporaryFile(delete=False) as tmp:
86      |     tmp_filename = tmp.name
87  assert read_file(tmp_filename) == ""
88  os.remove(tmp_filename)
89

```

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
File not found.
❖ saivenkatesh@Sais-MacBook-Air LAB Assignments %

```

EXPLANATION:

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 user-friendly 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 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:

#Detect and fix the error in this Python code where a non-existent 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 start(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:

Car is driving

```

A7.1.py > ...
94         return "Car started"
95     my_car = Car()
96     print(my_car.drive()) # drive() is not defined
97 # The error in the code is that the method drive() is called on the Car object, but it is not defi
98 class Car:
99     def start(self):
100         return "Car started"
101
102     def drive(self):
103         return "Car is driving"
104 my_car = Car()
105 print(my_car.drive())
106 # Assert tests
107 assert my_car.start() == "Car started"
108 assert my_car.drive() == "Car is driving"
109 another_car = Car()
110 assert another_car.start() == "Car started"
111 assert another_car.drive() == "Car is driving"
112

```

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/LAB Assignments/A7.1.py"
Car is driving

```

```

saivenkatesh@Sais-MacBook-Air LAB Assignments %

```

EXPLANATION:

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 `AttributeError`. Python cannot execute a method that does not exist in the class. The issue is fixed by defining the missing `drive()` method (or correcting the method call), allowing the object to use the method successfully.

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:

#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:

```
def add_five_numeric(value):  
    return int(value) + 5  
print(add_five_numeric("10"))  
# Assert tests for numeric addition  
assert add_five_numeric("10") == 15  
assert add_five_numeric("0") == 5  
assert add_five_numeric("-5") == 0  
# 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") == "Hello5"  
assert add_five_string("") == "5"
```

OUTPUT:

15
105

```
114 #detect and fix the TypeError in this Python code caused by adding a string and an integer:
115 #def add_five(value): return value + 5 ; print(add_five("10"))
116 #Explain why the error occurs, provide two solutions (one using type casting for numeric addition
117 # The error occurs because the code is trying to add an integer (5) to a string ("10"), which is
118 # Solution 1: Using type casting for numeric addition
119 def add_five_numeric(value):
120     return int(value) + 5
121 print(add_five_numeric("10"))
122 # Assert tests for numeric addition
123 assert add_five_numeric("10") == 15
124 assert add_five_numeric("0") == 5
125 assert add_five_numeric("-5") == 0
126 # Solution 2: Using string concatenation
127 def add_five_string(value):
128     return value + "5"
129 print(add_five_string("10"))
130 # Assert tests for string concatenation
131 assert add_five_string("10") == "105"
132 assert add_five_string("Hello") == "Hello5"
133 assert add_five_string("") == "5"
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
```

Explanation:

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.

