Lab 7: Error Debugging with AI

By- K.Sidhartha Reddy [2403A51293]

Title: Systematic Approaches to Finding and Fixing Bugs

**Task 1: Syntax Errors – Missing Parentheses in Print Statement**

**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**

• 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.

**Given Buggy Code**

Bug: Missing parentheses in print statement

def greet():  
 print "Hello, AI Debugging Lab!"  
greet()

**Observed Error**

**Error Message:**

File "main.py", line 3  
 print "Hello, AI Debugging Lab!"  
 ^^^^^^^^^^^^^^^^^^^^^^^^^  
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?

**Cause of Error:**

• The print statement is using Python 2 syntax.  
• In Python 3, print must be used as a function with parentheses.

**AI-Suggested Fix**

• Change print "text" to print("text").  
• Optionally make the function return the string to allow testing with assertions.

**Corrected Code with Assert Test Cases**

def greet():  
 # Fixed: added parentheses to print syntax  
 return "Hello, AI Debugging Lab!"  
  
Call the function and print the result  
print(greet())  
  
Assert test cases to confirm correct output  
assert greet() == "Hello, AI Debugging Lab!"  
assert isinstance(greet(), str)  
assert "AI Debugging" in greet()

Expected Output

Hello, AI Debugging Lab!  
All assert test cases pass successfully.

**Conclusion**

This task demonstrated how AI can help detect syntax errors, explain why they occur, and provide corrected code. Using AI suggestions allowed us to fix the missing parentheses quickly and verify the fix using assert-based test cases.

**Task 2: Logic Error – Incorrect Calculation**

**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**

• 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.

**Given Buggy Code**

# Bug: Logic error - incorrect area calculation  
def area\_of\_rectangle(length, width):  
 return length + width  
  
print(area\_of\_rectangle(5, 3))

**Observed Error / Issue**

Output: 8 (Expected 15)

Cause of Error:

Used + instead of \* for area calculation.

**AI-Suggested Fix**

Replace + with \* to correctly calculate area.

**Corrected Code with Assert Test Cases**

def area\_of\_rectangle(length, width):  
 return length \* width  
  
print(area\_of\_rectangle(5, 3))  
  
assert area\_of\_rectangle(5, 3) == 15  
assert area\_of\_rectangle(2, 4) == 8  
assert area\_of\_rectangle(10, 1) == 10

**Expected Output**

15  
All asserts pass.

**Conclusion**

AI helped detect a logic error where the wrong operator was used, suggested the correct operator, and fixed the logic.

**Task #3: Runtime Error – Division by Zero**

**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**

• 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.

**Given Buggy Code**

# Bug: Runtime error - division by zero  
def divide(a, b):  
 return a / b  
  
print(divide(10, 0))

**Observed Error / Issue**

ZeroDivisionError: division by zero

Cause of Error:

Dividing by zero is not allowed.

**AI-Suggested Fix**

Add condition to check if b == 0 before dividing.

**Corrected Code with Assert Test Cases**

def divide(a, b):  
 if b == 0:  
 return "Error: Cannot divide by zero"  
 return a / b  
  
print(divide(10, 2))  
print(divide(10, 0))  
  
assert divide(10, 2) == 5  
assert divide(9, 3) == 3  
assert divide(10, 0) == "Error: Cannot divide by zero"

**Expected Output**

5  
Error: Cannot divide by zero  
All asserts pass.

**Conclusion**

AI explained the cause of the runtime error and suggested adding a check to prevent division by zero.

**Task #4: Name Error – Undefined Variable**

**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**

• 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.

**Given Buggy Code**

# Bug: Name error - undefined variable  
def show\_name():  
 print(username)  
  
show\_name()

**Observed Error / Issue**

NameError: name 'username' is not defined

Cause of Error:

Variable username is not defined before use.

**AI-Suggested Fix**

Define the variable before using it or pass as parameter.

**Corrected Code with Assert Test Cases**

def show\_name(username):  
 return username  
  
print(show\_name("Alice"))  
  
assert show\_name("Alice") == "Alice"  
assert show\_name("Bob") == "Bob"  
assert isinstance(show\_name("Charlie"), str)

**Expected Output**

Alice  
All asserts pass.

**Conclusion**

AI identified the NameError and suggested defining or passing the variable as a parameter.

**Task #5: Type Error – Incompatible Types**

**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**

• 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.

**Given Buggy Code**

# Bug: Type error - adding number and string  
def add\_numbers(a, b):  
 return a + b  
  
print(add\_numbers(5, "3"))

**Observed Error / Issue**

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Cause of Error:

Cannot add an integer and string directly.

**AI-Suggested Fix**

Convert string to int before addition if it is numeric.

**Corrected Code with Assert Test Cases**

def add\_numbers(a, b):  
 if isinstance(b, str):  
 b = int(b)  
 return a + b  
  
print(add\_numbers(5, "3"))  
  
assert add\_numbers(5, "3") == 8  
assert add\_numbers(2, 4) == 6  
assert isinstance(add\_numbers(1, "9"), int)

**Expected Output**

8  
All asserts pass.

**Conclusion**

AI helped detect a TypeError and sugge