**HALL TICKET NUMBER: 2403A51365**

**NAME: RAYAPUDI HARITHA**

**BATCH: 24BTCAICSB14**

**AssignmentNumber:7.1**

**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.  
Week4 -  
Monday**

**• 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**

**Prompt:-**

**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.**

**Code And Output:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code Explanation:-**

**This Python code defines a simple function called greet. Inside the function, the line print("Hello, AI Debugging Lab!") is executed. This line uses the built-in Python print() function to display the string "Hello, AI Debugging Lab!" to the console. Finally, the line greet() calls the defined function, causing the message to be printed. The comment # Fixed: Added parentheses to print statement indicates a previous syntax error was corrected by adding parentheses around the string in the print function.**

**Task Description #2 (Logic Error – 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:-**

**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.**

**Code And Output:-**

A screenshot of a computer

AI-generated content may be incorrect.

**Code Explanation:-**

This Python code defines a function check\_number(n) that takes an integer n as input. It uses an if statement to compare n with the value 10. If n is equal to 10, the function returns the string "Ten"; otherwise, it returns "Not Ten". The code then includes three assert statements to test the function with different inputs (10, 5, and 15). If all assertions pass, it prints "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.**

**Prompt:-**

**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.**

**Code:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**OutPut:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code Explanation:-**

**This code defines a function read\_file\_safely to read file content. It uses a try-except block for robust error handling. The try part attempts to open and read the file. If the file isn't found, a FileNotFoundError is caught, and a specific error message is returned. A broader except Exception handles other potential issues. The code then tests this function with scenarios for an existing file, a missing file, and an invalid path (like a directory), printing informative messages for each outcome.**

**Task Description #4 (AttributeError – 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**

**Prompt:-  
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.**

**Code And Output:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code Explanation:-**

**This code defines a simple Python class named Car. Inside the Car class, there is one method defined: start(self). This start method, when called, simply returns the string "Car started". An instance of the Car class is then created and assigned to the variable my\_car. The start() method of the my\_car object is called, and its return value is stored in the result variable, which is then printed. Following this, the code includes three assert statements to test the functionality. The first assert checks if calling my\_car.start() returns the expected string "Car started". The second assert verifies that the my\_car object is indeed an instance of the Car class. The third assert checks if the return value of the start() method is of type string. Finally, if all these tests pass without raising an AssertionError, the message "All test cases passed!" is printed.**

**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.**

**Prompt:-**

**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.**

**Code And Output:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code Explanation:-**

**This code demonstrates two ways to handle potential TypeError when combining strings and integers. The add\_five\_type\_casting function converts the input value to an integer using int() before adding 5. It includes error handling for invalid conversions. The add\_five\_string\_concat function converts the integer 5 to a string using str() and then concatenates it with the input value. The code then provides separate sets of assert statements to test each function. The type casting tests check for valid integer inputs (as strings or integers) and an invalid string input. The string concatenation tests check for string inputs, demonstrating how '5' is appended. Finally, messages are printed to indicate if the tests for each solution passed. This illustrates two distinct approaches to handling mixed data types based on the desired outcome: mathematical addition or string joining.**