ASSIGNMENT - 7.3

NAME: K.ABHINAY

ROLL NO. : 2303A51899

BATCH NO. : 09

QuestioN:

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 -
Wednesday
• 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 for 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 using AI-generated insights
• Refactor buggy code using reliable programming patterns

TASK _ 1:

Fixing Syntax Errors
Scenario
You are reviewing a Python program where a basic function definition contains a syntax error.
Requirements
• Provide a Python function add(a, b) with a missing colon
• Use an AI tool to detect the syntax error
• Allow AI to correct the function definition

• Observe how AI explains the syntax issue
Expected Output
• Corrected function with proper syntax
• Syntax error resolved successfully
• AI-generated explanation of the fix

TASK _ 2:

Debugging Logic Errors in Loops
Scenario
You are debugging a loop that runs infinitely due to a logical mistake.
Requirements
• Provide a loop with an increment or decrement error
• Use AI to identify the cause of infinite iteration
• Let AI fix the loop logic
• Analyze the corrected loop behavior
Expected Output
• Infinite loop issue resolved
• Correct increment/decrement logic applied
• AI explanation of the logic error

TASK _ 3:

Handling Runtime Errors (Division by Zero)
Scenario
A Python function crashes during execution due to a division by zero error.
Requirements
• Provide a function that performs division without validation
• Use AI to identify the runtime error
• Let AI add try-except blocks for safe execution
• Review AI's error-handling approach
Expected Output
• Function executes safely without crashing
• Division by zero handled using try-except
• Clear AI-generated explanation of runtime error handling

TASK _ 4:

Debugging Class Definition Errors
Scenario
You are given a faulty Python class where the constructor is incorrectly defined.
Requirements
• Provide a class definition with missing self-parameter
• Use AI to identify the issue in the __init__() method
• Allow AI to correct the class definition
• Understand why self is required
Expected Output

• Corrected __init__() method
• Proper use of self in class definition
• AI explanation of object-oriented error

TASK _ 5:

Resolving Index Errors in Lists
Scenario
A program crashes when accessing an invalid index in a list.
Requirements
• Provide code that accesses an out-of-range list index
• Use AI to identify the Index Error
• Let AI suggest safe access methods
• Apply bounds checking or exception handling
Expected Output
• Index error resolved
• Safe list access logic implemented
• AI suggestion using length checks or exception handling

CODE:

```python
#Write a Python program that demonstrates and fixes common programming
errors including syntax error, logic error in loops, runtime error
(division by zero), class initialization error, and index error in lists
using proper debugging and exception handling techniques.
def task1_syntax_error():
    print("Task 1: Fixing Syntax Errors")
    buggy = "def add(a, b)\n    return a + b\n"
    print("Buggy code (string)")
    print(buggy.rstrip())
    try:
        exec(buggy, {})
        print("Buggy code executed (unexpected).")
    except SyntaxError as e:
        print(f"Detected SyntaxError: {e.msg} (line {e.lineno})")
        print("Explanation: Missing colon at end of function header. Fix:
add ':' -> def add(a, b):")
    def add(a, b):
        return a + b
    print("Corrected code")
    print("add(2, 3) ->", add(2, 3))
    print()

def task2_logic_error_loop():
    print("Task 2: Debugging Logic Errors in Loops")
    buggy = """def countdown(n):
    while n >= 0:
        print(n)
        n += 1
"""
    print("Buggy code (string)")
    print(buggy.rstrip())
    print("Issue: The loop increments `n` making it grow, so the condition
`n >= 0` never becomes False -> infinite loop.")
    print("Fix: Decrement `n` (n -= 1) or change loop condition.")
    def countdown(n):
        while n >= 0:
            print(n)
            n -= 1
    print("Corrected output (countdown from 3)")
    countdown(3)
    print()

def task3_runtime_divide_by_zero():
    print("Task 3: Handling Runtime Errors (Division by Zero)")
    buggy_example = "def divide(a, b):\\n    return a /
b\\nprint(divide(10, 0))"
    print("Buggy snippet ")
    print(buggy_example)
    print("Issue: Division by zero raises ZeroDivisionError at runtime.")
    def divide_safe(a, b):
        try:
            return a / b
        except ZeroDivisionError:
            return None
    print("Corrected output")
    print("divide_safe(10, 2) ->", divide_safe(10, 2))
    print("divide_safe(10, 0) ->", divide_safe(10, 0), "(None indicates
division by zero handled)")
    print()
```

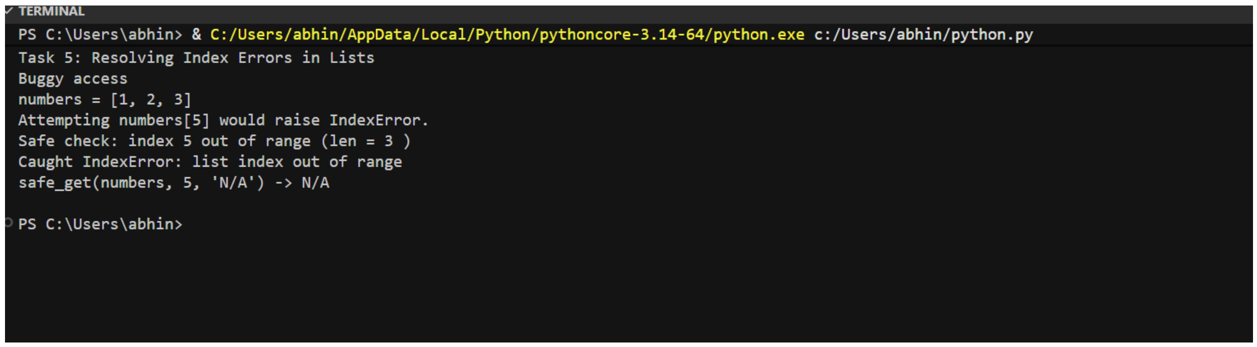
```
def task4_class_init_error():
    print("Task 4: Debugging Class Definition Errors")
    buggy = """class rectangle:
    def __init__(width, height):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
r = rectangle(4, 5)
"""
    print("Buggy code (string)")
    print(buggy.rstrip())
    print("Issues:")
    print("- __init__ missing `self` parameter; first parameter must be
`self`.")
    print("- Uses `length` which is undefined; parameter name mismatch.")
    print("Fix: def __init__(self, length, width): self.length = length;
self.width = width")
    class Rectangle:
        def __init__(self, length, width):
            self.length = length
            self.width = width
        def area(self):
            return self.length * self.width
    r = Rectangle(4, 5)
    print("Corrected usage")
    print("Rectangle(4,5).length ->", r.length)
    print("Rectangle(4,5).width  ->", r.width)
    print("Rectangle area ->", r.area())
    print()

def task5_index_error():
    print("Task 5: Resolving Index Errors in Lists")
    numbers = [1, 2, 3]
    print("Buggy access")
    print("numbers =", numbers)
    print("Attempting numbers[5] would raise IndexError.")
    idx = 5
    if 0 <= idx < len(numbers):
        print("numbers[5] ->", numbers[idx])
    else:
        print("Safe check: index 5 out of range (len =", len(numbers),
")")
    try:
        print("Try/except access result:", numbers[5])
    except IndexError as e:
        print("Caught IndexError:", str(e))
    def safe_get(lst, i, default=None):
        return lst[i] if 0 <= i < len(lst) else default
    print("safe_get(numbers, 5, 'N/A') ->", safe_get(numbers, 5, 'N/A'))
    print()

def main():
    task1_syntax_error()
    task2_logic_error_loop()
    task3_runtime_divide_by_zero()
    task4_class_init_error()
    task5_index_error()

if __name__ == "__main__":
    main()
```

OUTPUT:



```
TERMINAL
PS C:\Users\abhin> & C:/Users/abhin/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/abhin/python.py
Task 5: Resolving Index Errors in Lists
Buggy access
numbers = [1, 2, 3]
Attempting numbers[5] would raise IndexError.
Safe check: index 5 out of range (len = 3 )
Caught IndexError: list index out of range
safe_get(numbers, 5, 'N/A') -> N/A

PS C:\Users\abhin>
```