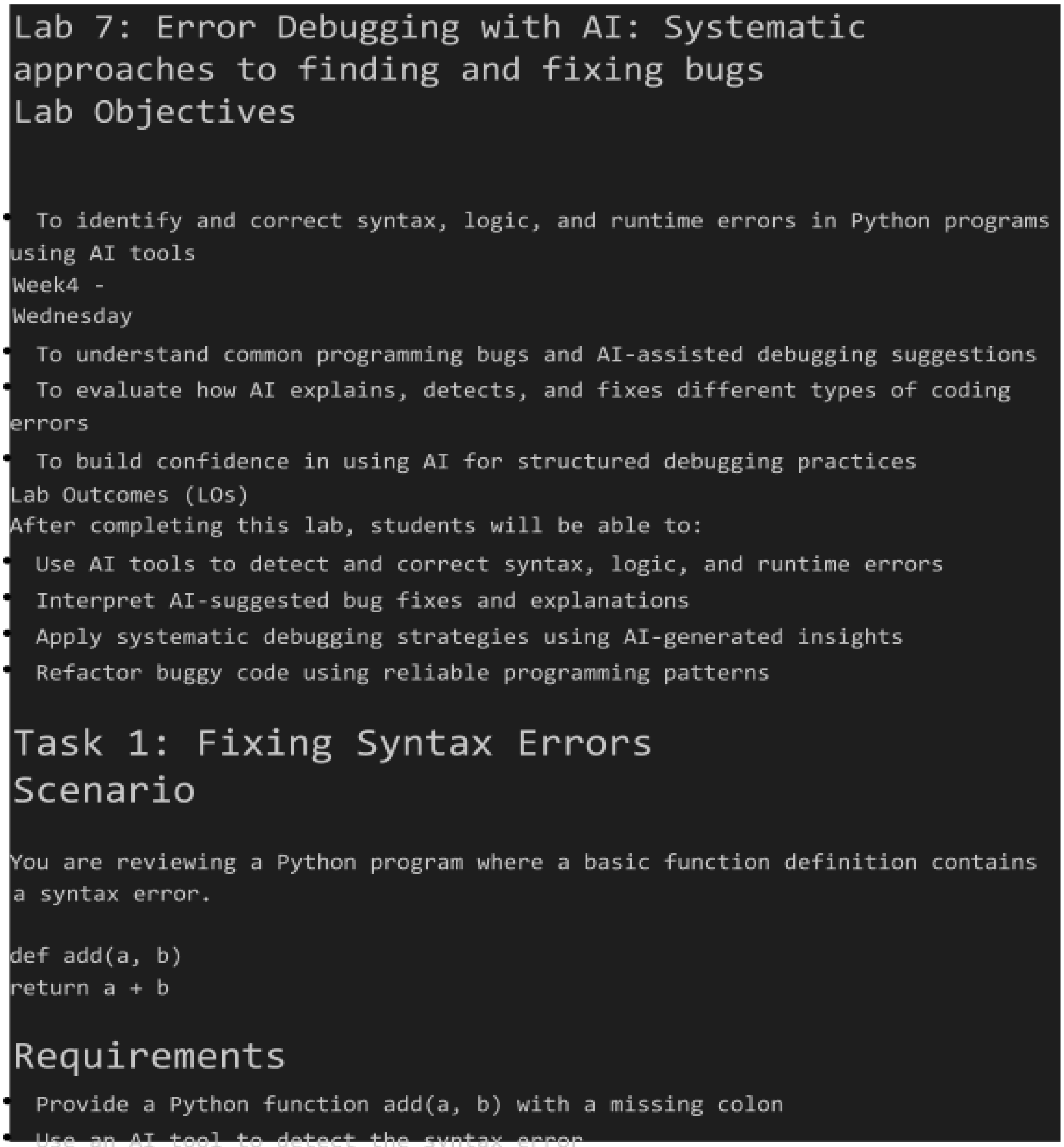
ASSIGNMENT – 7.3

Name : K. ABHINAY

Hall Ticket No : 2303A51899

Batch No. : 09



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.

```
def add(a, b)
return a + b
```

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

Prompt: Please inspect this Python snippet and act as an expert Python tutor:
Identify the syntax error in one sentence.
Provide the corrected function only (complete code block).
Provide a one-line test output for add(2, 3).
Reply with exactly those three items.

Code:

```python
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()
```

Output:

```
Task 1: Fixing Syntax Errors
--- Buggy code (string) ---
def add(a, b)
    return a + b
Detected SyntaxError: expected ':' (line 1)
Explanation: Missing colon at end of function header. Fix: add ':' ->
def add(a, b):
--- Corrected code ---
add(2, 3) -> 5

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

Task 2: Debugging Logic Errors in Loops

Scenario:

You are debugging a loop that runs infinitely due to a logical mistake.

```
def countdown(n):
    while n >= 0:
        print(n)
        n += 1
```

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

Prompt:

Find and fix infinite-loop logic

```
def countdown(n):
    while n >= 0:
        print(n)
        n += 1
```

Explain why this causes an infinite loop in one sentence.
Provide the corrected function only (complete code block) that
counts down from n to 0.
Show the printed output lines for countdown(3).

Code:

```
def task2_logic_error_loop():
    print("Task 2: Debugging Logic Errors in Loops")
```

```
buggy = """def countdown(n):
while n >= 0:
    p
    r
"""  i
    n
    t
    (
    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()
```

Output:

```
Task 3: Handling Runtime Errors (Division by Zero)
--- Buggy snippet ---
def divide(a, b):\n    return a / b\nprint(divide(10, 0))
Issue: Division by zero raises ZeroDivisionError at
runtime.
--- Corrected output ---
divide_safe(10, 2) -> 5.0
divide_safe(10, 0) -> None (None indicates division by zero handled)
```

Task 3: Handling Runtime Errors (Division by Zero)

Scenario:

A Python function crashes during execution due to a division by zero error.

Requirements

```
def divide(a, b):
    return a / b
print(divide(10, 0))
```

- 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

Prompt:

Handle division-by-zero runtime error

Given this snippet:

```
def divide(a, b):
    return a / b
print(divide(10, 0))
Identify the runtime error in one sentence.
Provide a safe divide(a, b) implementation using try/except that returns a clear value or message on error.
Show example outputs for divide(10,2) and divide(10,0).
```

Code:

```python
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()
```

Output:

```
Task 4: Debugging Class Definition Errors
--- Buggy code (string) ---
class rectangle:
    def _init_(width, height):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
r = rectangle(4, 5)
Issues:
- _init__missing `self` parameter; first parameter must be `self`.
- Uses `length` which is undefined; parameter name mismatch.
```

```
Fix: def _init_(self, length, width): self.length = length; self.width =
width
--- Corrected usage ---
Rectangle(4,5).length -> 4
Rectangle(4,5).width -> 5
Rectangle area -> 20
```

Task 4: Debugging Class Definition Errors

Scenario

You are given a faulty Python class where the constructor is incorrectly defined.

Requirements

```
class rectangle:
    def _init_(width, height):
        self.length = length
        self.width = width
```

- 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

Prompt:

fix incorrect class constructor
Inspect this class:

```
class rectangle:
    def _init_(width, height):
        self.length = length
        self.width = width
```

Describe the problems in two short bullets.
Provide a corrected class definition (use Rectangle, proper self, correct parameters) with a small area() method.
Show sample usage and output for Rectangle(4,5).area()

Code:

```python
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()
```

Output:

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

```
Rectangle(4,5).length -> 4
Rectangle(4,5).width -> 5
Rectangle area -> 20
```

Task 5: Resolving Index Errors in Lists

Scenario

A program crashes when accessing an invalid index in a list.

Requirements

```python
numbers = [1, 2, 3]
print(numbers[5])
```

• 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

Prompt:

Resolve list index error safely
Given:

numbers = [1, 2, 3]

print(numbers[5])

Explain why this raises an error in one sentence.

Provide two safe alternatives: (a) bounds-check access, (b) try/except fallback returning a default. Show code for both.

Show example outputs for index 5 with both methods.

Code:

```python
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:

```
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
```

A hospital wants to implement a Patient Appointment and Prescription Record System to store patients visit details and generate dally medical reports. You are required to develop a Python program using classes, constructors, loops, conditional statements, and file handling. Each patent record should contain patient (D. patient name, age, doctor name, age, doctor name and diagnosis, and prescribed medicines. When a new patient visit is entered, the male must be stored permanently in a the. The program should allow the receptionist to repeatedly perform operations such as adding a new patient visit, viewing all patient recant searching for a patient by ID, and updating diagnosis. The system must also analyze potent visits by calculating the total number of patients vested in a day identifying the most frequently consulted doctor, and listing patients above 50 years of age for priority care. The program should continue running until the user chooses to eat.

Prompt:

A hospital wants to implement a Patient Appointment and Prescription Record System to store patients visit details and generate dally medical reports

Code:

```
import os

FILENAME = "patients.txt"

# Class Definition

class Patient:

    def _init_(self, patient_id, name, age, doctor, diagnosis,
medicines):

        self.patient_id = patient_id

        self.name = name

        self.age = age

        self.doctor = doctor

        self.diagnosis = diagnosis

        self.medicines = medicines



    def to_string(self):

        return
f"{self.patient_id},{self.name},{self.age},{self.doctor},{self.diagnosis},
{self.medicines}\n"

# Clear File (For Fresh Output)

open(FILENAME, "w").close()

# Adding Patients Directly
```

```
p1 = Patient("101", "Ravi", 60, "Dr. Sharma", "Fever", "Paracetamol")

p2 = Patient("102", "Anjali", 35, "Dr. Sharma", "Cold", "Cetirizine")

p3 = Patient("103", "Suresh", 55, "Dr. Reddy", "Diabetes", "Insulin")

with open(FILENAME, "a") as file:

    file.write(p1.to_string())

    file.write(p2.to_string())

    file.write(p3.to_string())

print("✅ Patient records added successfully!\n")

# View All Patients

print("---- All Patient Records     ")

with open(FILENAME, "r") as file:

    records = file.readlines()

    for record in records:

        data = record.strip().split(",")

        print(f"ID: {data[0]}, Name: {data[1]}, Age: {data[2]}, "

              f"Doctor: {data[3]}, Diagnosis: {data[4]}, Medicines:
{data[5]}")

print()

# Search Patient by ID

search_id = "101"

print(f"---- Searching for Patient ID {search_id}     ")


with open(FILENAME, "r") as file:

    for line in file:

        data = line.strip().split(",")

        if data[0] == search_id:
            print(f"Patient Found: {data[1]}, Age: {data[2]}, Doctor:
{data[3]}\n")

# Update Diagnosis

update_id = "102"

new_diagnosis = "Migraine"

lines = []

with open(FILENAME, "r") as file:

    lines = file.readlines()

with open(FILENAME, "w") as file:

    for line in lines:

        data = line.strip().split(",")

        if data[0] == update_id:

            data[4] = new_diagnosis

            print(f"✅ Diagnosis updated for Patient ID {update_id}")

        file.write(",".join(data) + "\n")

print()

# Total Patients

with open(FILENAME, "r") as file:

   total = len(file.readlines())

print(f"D Total Patients Visited Today: {total}\n")

# - Most Frequently Consulted Doctor

doctor_count = {}

with open(FILENAME, "r") as file:

    for line in file:

        data = line.strip().split(",")

        doctor = data[3]

        doctor_count[doctor] = doctor_count.get(doctor, 0) + 1
```

```
most_doctor = max(doctor_count, key=doctor_count.get) print(f"

👨‍⚕️ Most Frequently Consulted Doctor: {most_doctor}\n") #

Patients Above 50

print("👴 Patients Above 50 Years:")

with open(FILENAME, "r") as file:

    for line in file:

        data = line.strip().split(",")

        if int(data[2]) > 50:

            print(f"ID: {data[0]}, Name: {data[1]}, Age: {data[2]}")
```