Assignment

Htno : 2503A52L16

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.

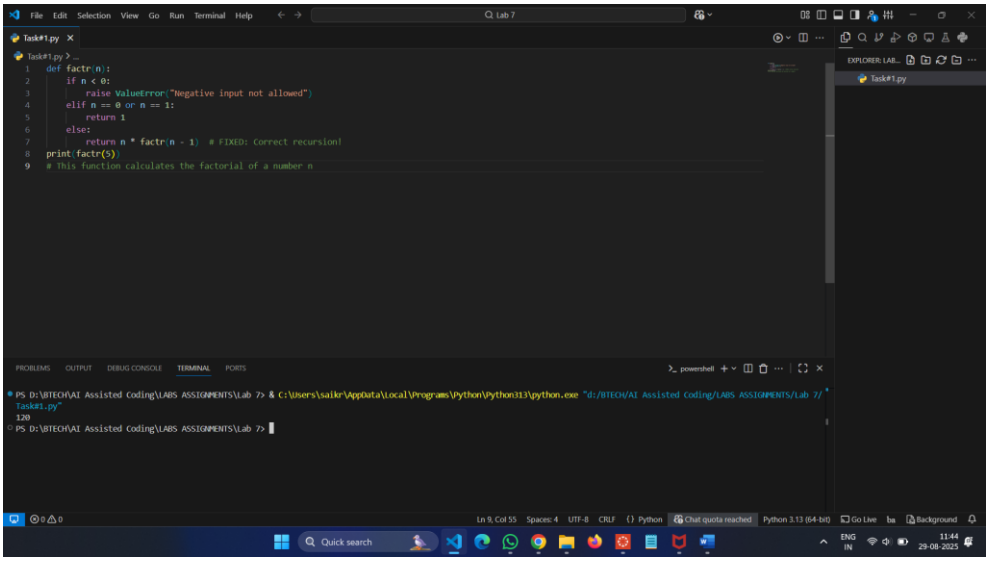
1. **Task Description #1:**

   Introduce a buggy Python function that calculates the factorial of a number using recursion. Use Copilot or Cursor AI to detect and fix the logical or syntax errors.

```
def factr(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return n * factr(n - 2)

print(factr("5"))
```

Expected Outcome #1:

- Copilot or Cursor AI correctly identifies missing base condition or incorrect recursive call and suggests a functional factorial implementation.



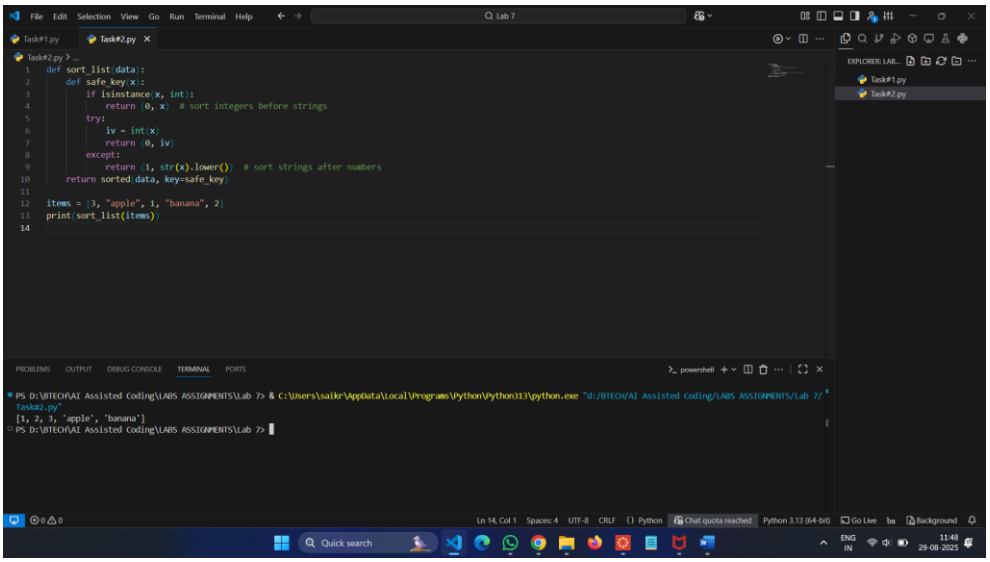
Task Description #2:

- Provide a list sorting function that fails due to a type error (e.g., sorting list with mixed integers and strings). Prompt AI to detect the issue and fix the code for consistent sorting**.**

```
def sort_list(data):
    return sorted(data)

items = [3, "apple", 1, "banana", 2]
print(sort_list(items))
```

Expected Outcome #2:

- AI detects the type inconsistency and either filters or converts list elements, ensuring successful sorting without a crash.



Task Description #3:

- Write a Python snippet for file handling that opens a file but forgets to close it. Ask Copilot or Cursor AI to improve it using the best practice (e.g., with open() block).

Code1

```
with open("example.txt", "w") as f:
    f.write("Hello, world!")
```

Code2

```
f1 = open("data1.txt", "w")
f2 = open("data2.txt", "w")

f1.write("First file content\n")
f2.write("Second file content\n")

print("Files written successfully")
```

Code3

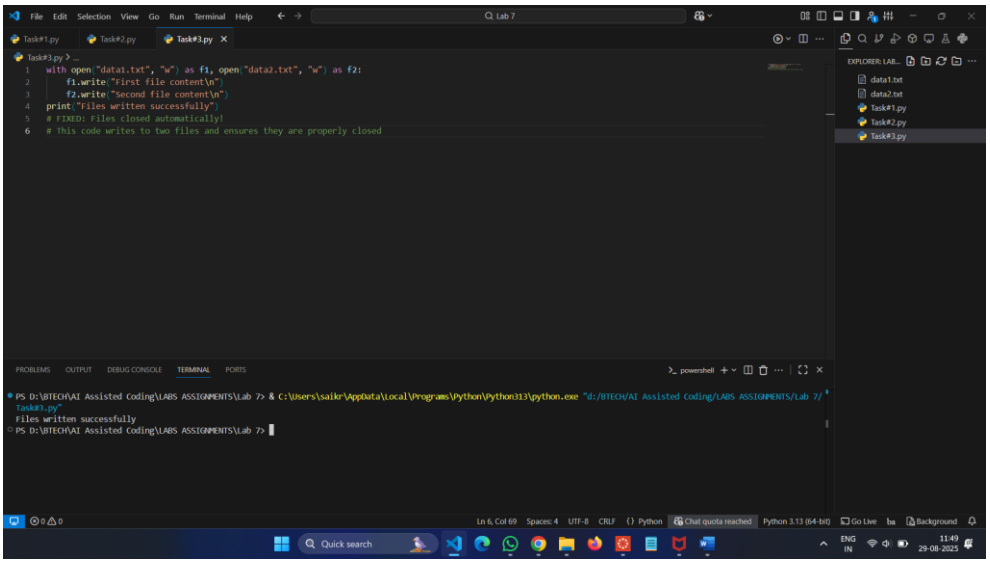
```
data = open("input.txt", "r").readlines()
output = open("output.txt", "w")

for line in data:
    output.write(line.upper())

print("Processing done")
```

Code4:

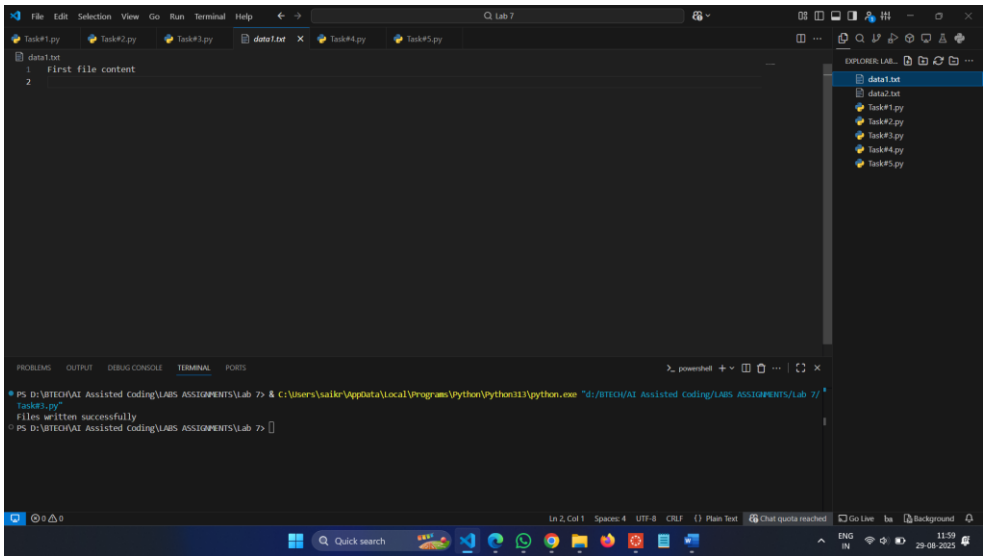
```
f = open("numbers.txt", "r")
nums = f.readlines()

squares = []
for n in nums:
    n = n.strip()
    if n.isdigit():
        squares.append(int(n) * int(n))

f2 = open("squares.txt", "w")
for sq in squares:
    f2.write(str(sq) + "\n")

print("Squares written")
```

Expected Outcome #3:

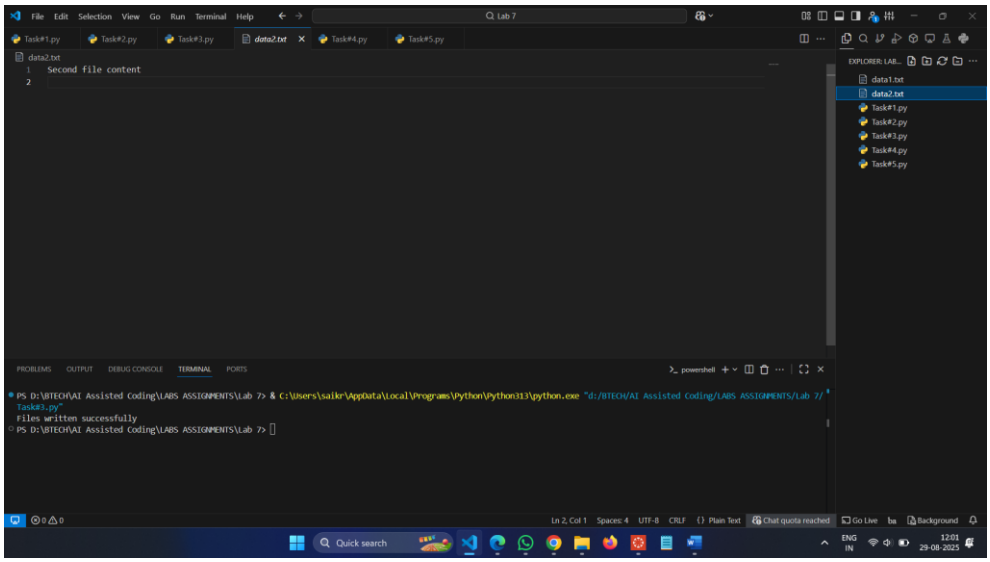
- AI refactors the code to use a context manager, preventing resource leakage and runtime warnings.



- **File one :**



- **File Two :**



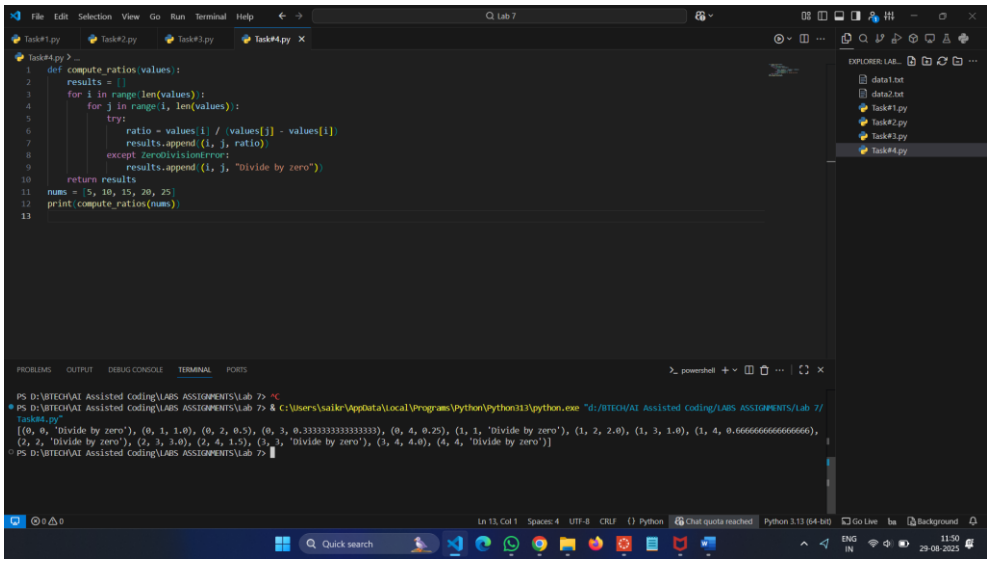
Task Description #4:

- Provide a piece of code with a ZeroDivisionError inside a loop. Ask AI to add error handling using try-except and continue execution safely.

```python
def compute_ratios(values):
    results = []
    for i in range(len(values)):
        for j in range(i, len(values)):
            ratio = values[i] / (values[j] - values[i])
            results.append((i, j, ratio))
    return results

nums = [5, 10, 15, 20, 25]
print(compute_ratios(nums))
```

Expected Outcome #4:

- Copilot adds a try-except block around the risky operation, preventing crashes and printing a meaningful error message.



Task Description #5:

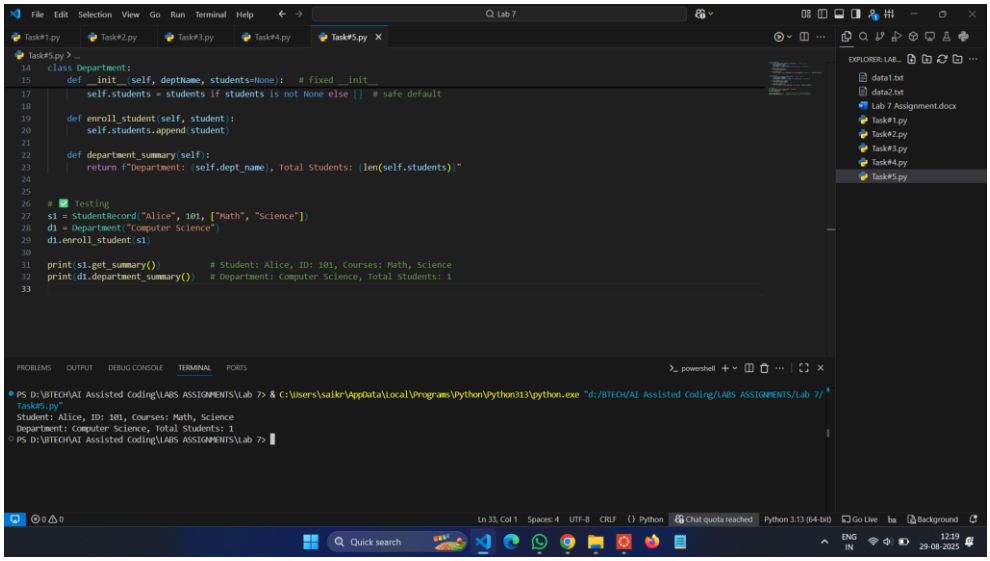
- Include a buggy class definition with incorrect __init__ parameters or attribute references. Ask AI to analyze and correct the constructor and attribute usage.

```
class StudentRecord:
    def __init__(self, name, id, courses=[]):
        self.studentName = names
        self.student_id = id
        self.courses = courseList

    def add_course(self, course):
        self.courses.append(course)

    def get_summary(self):
        return f"Student: {self.studentName}, ID: {self.student_id}, Courses: {',
'.join(self.courses)}"
```

```
class Department:
    def __init__(self, deptName, students=None):
        self.dept_name = deptName
        self.students = students
    def enroll_student(self, student):
        self.students.append(student)

    def department_summary(self):
        return f"Department: {self.dept_name}, Total Students: {len(self.student)}
s1 = StudentRecord("Alice", 101, ["Math", "Science"])
d1 = Department("Computer Science")
d1.enroll_student(s1)
print(s1.get_summary())
print(d1.department_summary())
```

Expected Outcome #5:

- Copilot identifies mismatched parameters or missing self references and rewrites the class with accurate initialization and usage.



- ## Observation

  Overall, the assignment reinforces core programming principles: correctness in algorithms, strict data handling, managing system resources carefully, robust error handling, safe object-oriented design, and the benefits of AI-supported debugging.
  This comprehensive approach prepares developers to write reliable, maintainable, and professional Python code.