SCHOOLOFCOMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENTOFCOMPUTER SCIENCE ENGINEERING	
ProgramName: B. Tech		**AssignmentType: Lab**	**AcademicYear:** 2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
Instructor(s)Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
DateandDay of Assignment	Week4 - Thursday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber: **7.4**(Presentassignmentnumber)/**24**(Totalnumberofassignments)			

Q.No.	Question	*ExpectedTime to complete*
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 - Thursday

- 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:

• 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.

Code:

```
def factr(n):
    if n == 0:
        return 1
    else:
        return n * factr(n - 1)
print(factr(5))
```

Output:

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.

Code:

```
def sort_list(data):
    return sorted(data, key=str)
items=[3,"apple",1,"banana",2]
print(sort_list(items))
```

Output:

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")
```

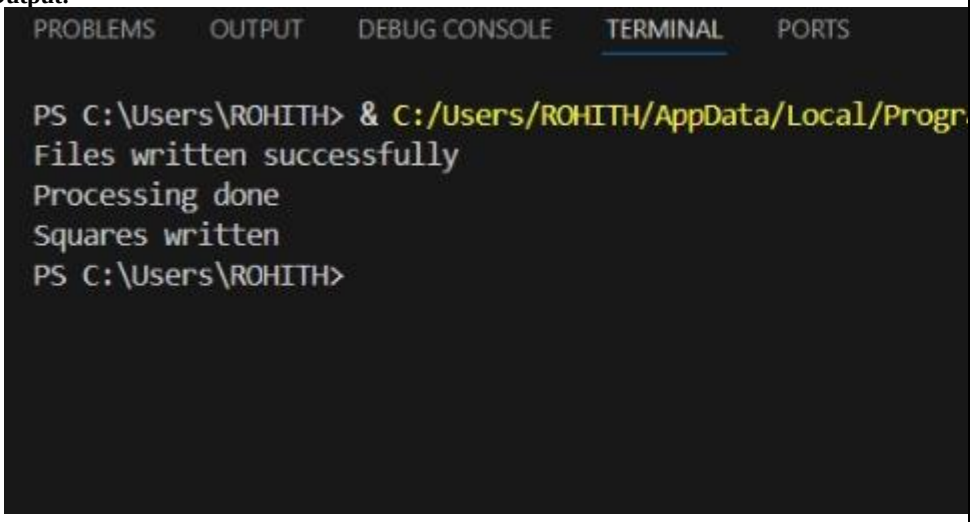
Expected Outcome #3:

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

```
import os

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

with open("data1.txt", "w") as f1, open("data2.txt", "w") as f2:
    f1.write("First file content\n")
    f2.write("Second file content\n")

print("Files written successfully")


# Create input.txt with sample data if it does not exist
if not os.path.exists("input.txt"):
    with open("input.txt", "w") as f:
```

```
        f.write("Hello\nWorld\nPython\nFile Handling\n")

# Create numbers.txt with sample numeric data if it does not exist
if not os.path.exists("numbers.txt"):
    with open("numbers.txt", "w") as f:
        f.write("1\n2\n3\n4\n5\nabc\n6\n")

with open("input.txt", "r") as data, open("output.txt", "w") as output:
    for line in data:
        output.write(line.upper())

print("Processing done")

with open("numbers.txt", "r") as f:
    nums = f.readlines()

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

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

print("Squares written")
```

Output:



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.

Code:

```python
def compute_ratios(values):
    results = []
    # ...existing code...
    for i in range(len(values)):
        for j in range(i, len(values)):
            try:
                ratio = values[i] / (values[j] - values[i])
                results.append((i, j, ratio))
            except Exception as e:
                print(f"Error computing ratio for i={i}, j={j}: {e}")
    # ...existing code...
    return results

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

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

PS C:\Users\ROHITH> & C:/Users/ROHITH/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ROHITH/OneDrive/Desktop/Rohith/zerodivision.py
Error computing ratio for i=0, j=0: division by zero
Error computing ratio for i=1, j=1: division by zero
Error computing ratio for i=2, j=2: division by zero
Error computing ratio for i=3, j=3: division by zero
Error computing ratio for i=4, j=4: division by zero
[(0, 1, 1.0), (0, 2, 0.5), (0, 3, 0.3333333333333333), (0, 4, 0.25), (1, 2, 2.0), (1, 3, 1.0), (1, 4, 0.6666666666666666), (2, 3, 3.0), (2, 4, 1.5), (3, 4
, 4.0)]
```

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.

Code:

```
class StudentRecord:
    def __init__(self, name, student_id, courses=None):
        self.studentName = name
        self.student_id = student_id
        self.courses = courses if courses is not None else []

    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 if students is not None else []

    def enroll_student(self, student):
        self.students.append(student)

    def department_summary(self):
        return f"Department: {self.dept_name}, Total Students: {len(self.students)}"

s1 = StudentRecord("Alice", 101, ["Math", "Science"])
d1 = Department("Computer Science")
d1.enroll_student(s1)
print(s1.get_summary())
print(d1.department_summary())
```

Output: