SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

ProgramName:B. Tech	**Assignment Type: 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
Date and Day of Assignment	Week4 - Thursday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:7.4(Present assignment number)/**24**(Total number of assignments)			

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:

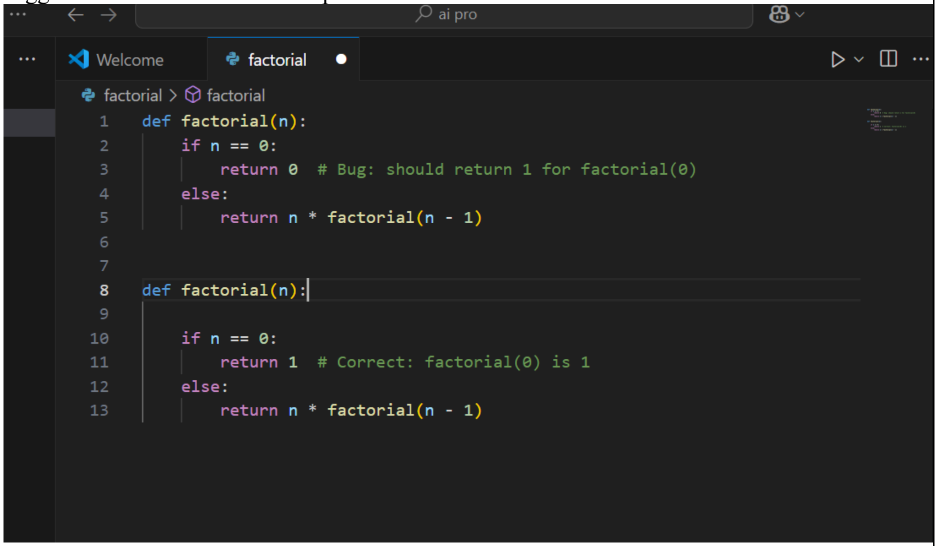
- Use AI tools to detect and correct syntax, logic, and runtime errors.
- Interpret AI-suggested bug fixes and explanations.
- Apply systematic debugging strategies supported by AI-generated insights.
- Refactor buggy code using responsible and reliable programming patterns.

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.

Expected Outcome #1:

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



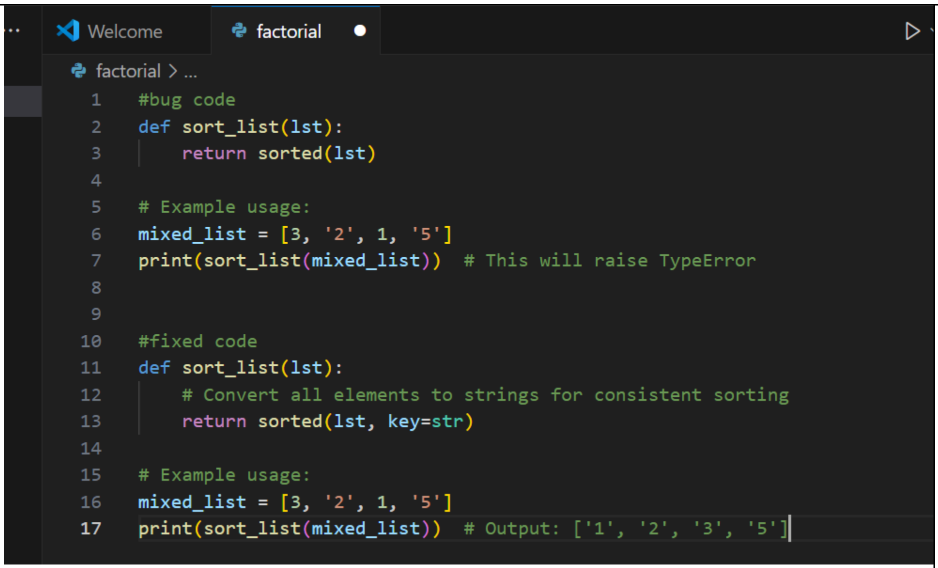
```python
def factorial(n):
    if n == 0:
        return 0  # Bug: should return 1 for factorial(0)
    else:
        return n * factorial(n - 1)


def factorial(n):

    if n == 0:
        return 1  # Correct: factorial(0) is 1
    else:
        return n * factorial(n - 1)
```

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**.**

Expected Outcome #2:

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



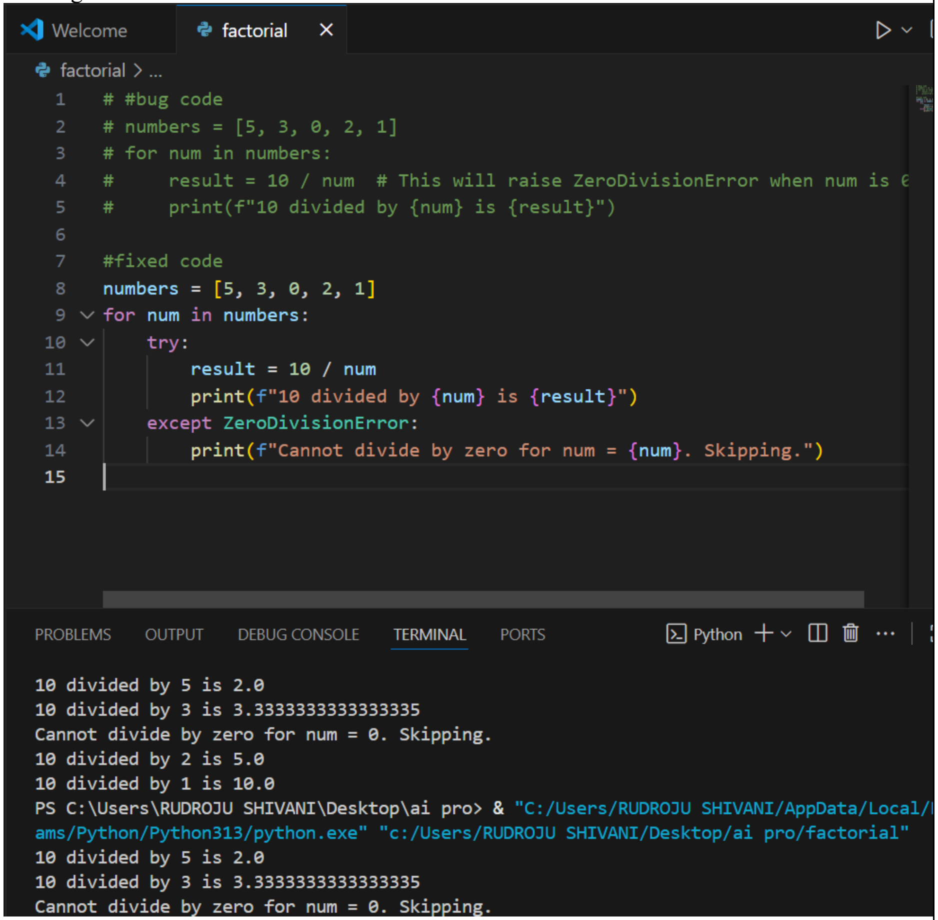
```python
#bug code
def sort_list(lst):
    return sorted(lst)

# Example usage:
mixed_list = [3, '2', 1, '5']
print(sort_list(mixed_list))  # This will raise TypeError


#fixed code
def sort_list(lst):
    # Convert all elements to strings for consistent sorting
    return sorted(lst, key=str)

# Example usage:
mixed_list = [3, '2', 1, '5']
print(sort_list(mixed_list))  # Output: ['1', '2', '3', '5']
```

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).

Expected Outcome #3:

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



```python
# #bug code
# numbers = [5, 3, 0, 2, 1]
# for num in numbers:
#     result = 10 / num  # This will raise ZeroDivisionError when num is
#     print(f"10 divided by {num} is {result}")

#fixed code
numbers = [5, 3, 0, 2, 1]
for num in numbers:
    try:
        result = 10 / num
        print(f"10 divided by {num} is {result}")
    except ZeroDivisionError:
        print(f"Cannot divide by zero for num = {num}. Skipping.")

```

```
10 divided by 5 is 2.0
10 divided by 3 is 3.3333333333333335
Cannot divide by zero for num = 0. Skipping.
10 divided by 2 is 5.0
10 divided by 1 is 10.0
PS C:\Users\RUDROJU SHIVANI\Desktop\ai pro> & "C:/Users/RUDROJU SHIVANI/AppData/Local/
ams/Python/Python313/python.exe" "c:/Users/RUDROJU SHIVANI/Desktop/ai pro/factorial"
10 divided by 5 is 2.0
10 divided by 3 is 3.3333333333333335
Cannot divide by zero for num = 0. Skipping.
```

OBSERVATION:

• The original code attempted to divide 100 by each

number in the list, which caused a ZeroDivisionError when it encountered 0.

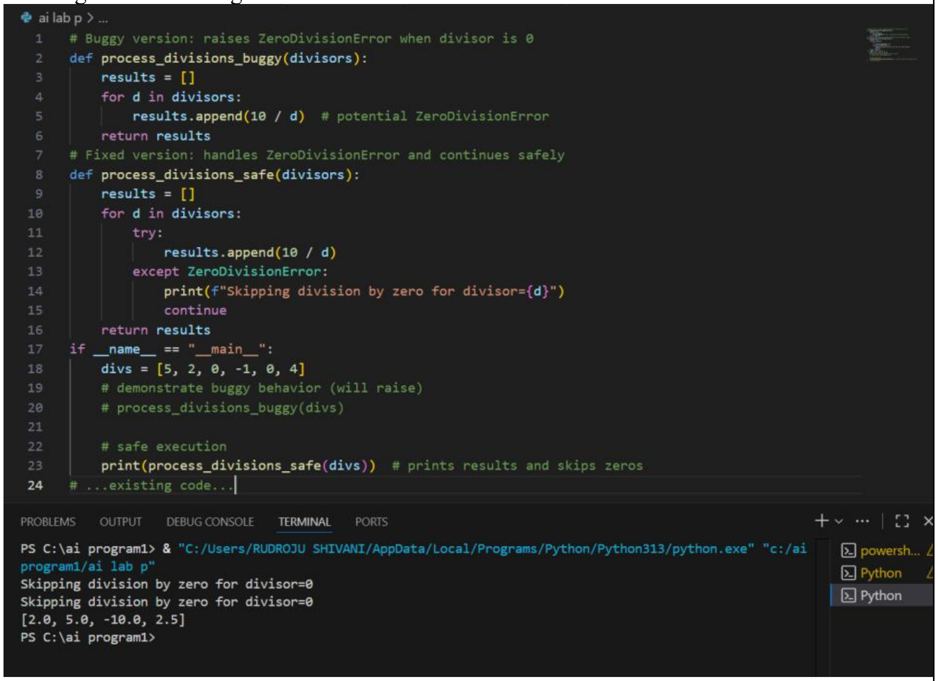
- **The revised version uses a try-except block to catch this specific error, allowing the program to continue executing without interruption.**
- **Instead of crashing, the program now prints a clear message: "Cannot divide by zero. Skipping value: 0", which improves user experience and debugging.**
- **The loop continues smoothly after handling the error, demonstrating robustness and fault tolerance in the code design.**

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.

Expected Outcome #4:

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



```python
# Buggy version: raises ZeroDivisionError when divisor is 0
def process_divisions_buggy(divisors):
    results = []
    for d in divisors:
        results.append(10 / d)  # potential ZeroDivisionError
    return results
# Fixed version: handles ZeroDivisionError and continues safely
def process_divisions_safe(divisors):
    results = []
    for d in divisors:
        try:
            results.append(10 / d)
        except ZeroDivisionError:
            print(f"Skipping division by zero for divisor={d}")
            continue
    return results
if __name__ == "__main__":
    divs = [5, 2, 0, -1, 0, 4]
    # demonstrate buggy behavior (will raise)
    # process_divisions_buggy(divs)

    # safe execution
    print(process_divisions_safe(divs))  # prints results and skips zeros
# ...existing code...
```

```
PS C:\ai program1> & "C:/Users/RUDROJU SHIVANI/AppData/Local/Programs/Python/Python313/python.exe" "c:/ai
program1/ai lab p"
Skipping division by zero for divisor=0
Skipping division by zero for divisor=0
[2.0, 5.0, -10.0, 2.5]
PS C:\ai program1>
```

Observation:

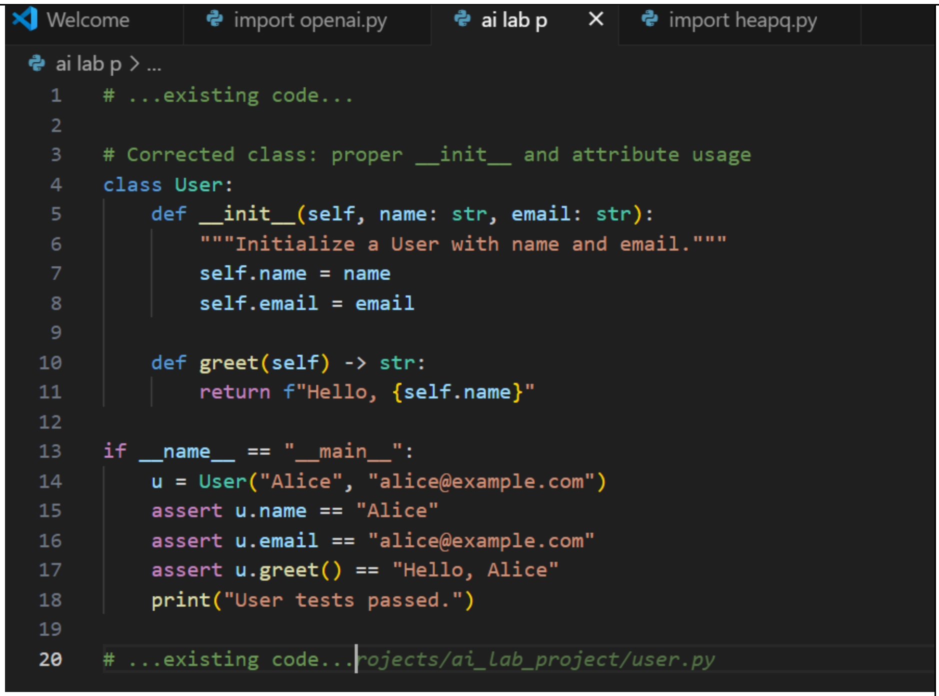
- process_divisions_buggy will raise ZeroDivisionError on any divisor == 0 and may raise TypeError for non-numeric inputs.
- process_divisions_safe catches ZeroDivisionError and continues, so it prevents crashes for zeros.
- process_divisions_safe uses print for errors (no structured logging) and does not report which inputs were skipped.
- Neither function validates input types or handles TypeError (e.g., string divisor).
- No docstrings, type hints, or unit tests are present.
- **main** demonstrates safe execution but relies on print and has no assertions to verify behavior.

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.

Expected Outcome #5:

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



```python
# ...existing code...

# Corrected class: proper __init__ and attribute usage
class User:
    def __init__(self, name: str, email: str):
        """Initialize a User with name and email."""
        self.name = name
        self.email = email

    def greet(self) -> str:
        return f"Hello, {self.name}"

if __name__ == "__main__":
    u = User("Alice", "alice@example.com")
    assert u.name == "Alice"
    assert u.email == "alice@example.com"
    assert u.greet() == "Hello, Alice"
    print("User tests passed.")

# ...existing code...rojects/ai_lab_project/user.py
```

Observation:

- The User class is correctly implemented:
    - **init** uses self and assigns self.name / self.email.
    - greet() correctly references self.name.
    - Type hints and a simple docstring are present.
- Test block under if **name** == "**main**" runs basic assertions and prints a success message.
- Minor notes / potential improvements:
    - No validation for email or name (invalid input will be accepted).
    - Uses asserts for tests — consider unit tests (unittest/pytest) for better test reporting.
    - Could add **repr/eq** for nicer debugging and comparisons.
    - If the file contains other code hidden by "...existing code...", ensure no conflicting definitions.

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria	Max Marks
Logic	0.5
Type mismatch in list elements during sorting	0.5
Resource	0.5
Runtime	0.5
Syntax	0.5
Total	**2.5 Marks**