| 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)<br>Dr. T. Sampath Kumar<br>Dr. Pramoda Patro<br>Dr. Brij Kishor Tiwari<br>Dr.J.Ravichander<br>Dr. Mohammand Ali Shaik<br>Dr. Anirodh Kumar<br>Mr. S.Naresh Kumar<br>Dr. RAJESH VELPULA<br>Mr. Kundhan Kumar<br>Ms. Ch.Rajitha<br>Mr. M Prakash<br>Mr. B.Raju<br>Intern 1 (Dharma teja)<br>Intern 2 (Sai Prasad)<br>Intern 3 (Sowmya)<br>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<br><br>Lab Objectives:<br>• 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"))
```

AFTER BUGGING:

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

**OUTPUT:**

```
ive/Desktop/AIAC/Lab-7/Task1.py
120
```

**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.

AFTER BUGGING:

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

OUTPUT:

```
ive/Desktop/AIAC/Lab-7/Task2.py
[1, 2, 3, 'apple', 'banana']
```

**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!")
```

**AFTER BUGGING:**

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

**OUTPUT:**

```
example.txt
1   Hello, World!
2
```

**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")
```

**AFTER BEGGING:**

```
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")
```

**OUTPUT:**

data1.txt

```
1    first file content
2
```

data2.txt

```
1    second file contect
2
```

**Code3**

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

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

print("Processing done")
```

**AFTER BUGGING:**

```python
with open("input.txt", "r") as data:
    lines = data.readlines()
with open("output.txt", "w") as output:
    for line in lines:
        output.write(line.upper())
print("Processing done")
```

**Code4:**

```python
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")
```

**AFTER BUGGING:**

```python
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.

**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.

AFTER BUGGING:

Task4.py > compute_ratios

```python
def compute_ratios(values):
    results = []
    for i in range(len(values)):
        for j in range(i, len(values)):
            if values[j] - values[i] != 0:
                ratio = values[i] / (values[j] - values[i])
                results.append((i, j, ratio))
            else:
                results.append((i, j, None))
    return results

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

OUTPUT:

```
ive/Desktop/AIAC/Lab-7/Task4.py
[(0, 0, None), (0, 1, 1.0), (0, 2, 0.5), (0, 3, 0.3333333333333333), (0, 4, 0.25), (1, 1, None), (1, 2, 2.0), (1, 3, 1.0), (1, 4, 0.666
6666666666666), (2, 2, None), (2, 3, 3.0), (2, 4, 1.5), (3, 3, None), (3, 4, 4.0), (4, 4, None)]
```

**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.

**AFTER BEGGING:**

Task5.py > StudentRecord

```
class StudentRecord:
    def __init__(self, name, id, courses=None):
        self.studentName = name
        self.student_id = id
        if courses is None:
            self.courses = []
        else:
            self.courses = courses

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

    def get_summary(self):
        return f"Student: {self.studentName}, ID: {self.studen

class Department:
    def __init__(self, deptName, students=None):
        self.dept_name = deptName
        if students is None:
            self.students = []
        else:
            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.students)}"


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

OUTPUT:

| | | | |
|---|---|---|---|
| | ive/Desktop/AIAC/Lab-7/Task3.py<br>Student: Alice, ID: 101, Courses: Math, Science<br>Department: Computer Science, Total Students: 1 | | |