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 - Wednesday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	

AssignmentNumber:7.3(Present assignment number)/**24**(Total number of assignments)

Q.No.	Question	*ExpectedTime to complete*
1	Lab 7: AI-Error Debugging with AI: Systematic approaches to finding and fixing bugs	Week4 - Wednesday

Lab Objectives:

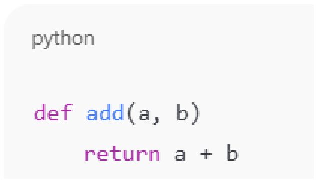
- To identify and correct syntax, logic, and runtime errors in Python programs using AI tools.
- To understand common programming bugs and AI-assisted debugging suggestions.
- To evaluate how AI explains, detects, and fixes different types of coding errors.
- To build confidence in using AI to perform structured debugging practices.

Lab Outcomes (LOs):
After completing this lab, students will be able to:

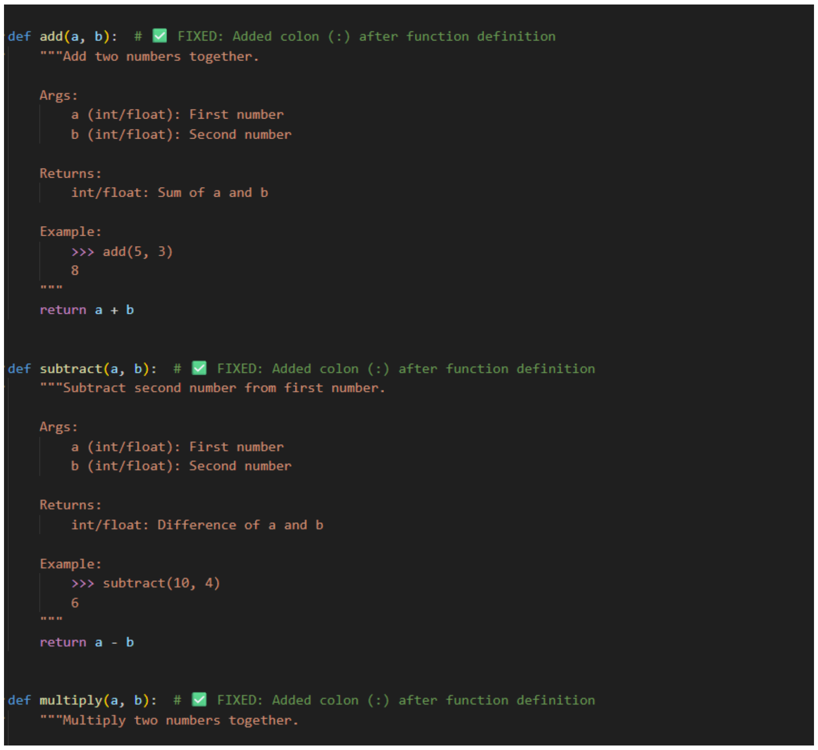
- Use AI tools to detect and correct syntax, logic, and runtime errors.
- Interpret AI-suggested bug fixes and explanations.
- Apply systematic debugging strategies supported by AI-generated insights.
- Refactor buggy code using responsible and reliable programming patterns.

Task Description#1

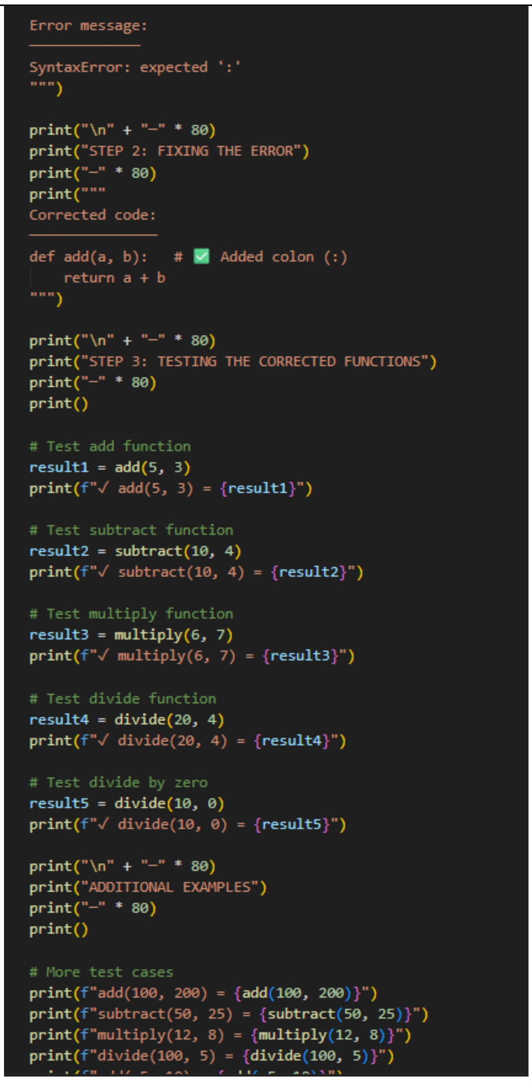
- Paste a function with a missing colon (add(a, b)), and let AI fix the syntax error.



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



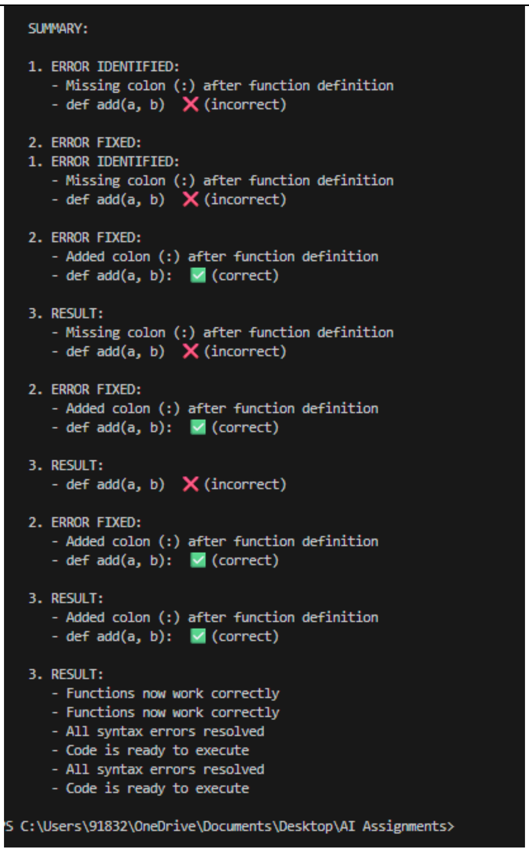
```python
def add(a, b):  # ✅ FIXED: Added colon (:) after function definition
    """Add two numbers together.

    Args:
        a (int/float): First number
        b (int/float): Second number

    Returns:
        int/float: Sum of a and b

    Example:
        >>> add(5, 3)
        8
    """
    return a + b


def subtract(a, b):  # ✅ FIXED: Added colon (:) after function definition
    """Subtract second number from first number.

    Args:
        a (int/float): First number
        b (int/float): Second number

    Returns:
        int/float: Difference of a and b

    Example:
        >>> subtract(10, 4)
        6
    """
    return a - b


def multiply(a, b):  # ✅ FIXED: Added colon (:) after function definition
    """Multiply two numbers together.
```



```
Error message:
─────────────
SyntaxError: expected ':'
""")

print("\n" + "─" * 80)
print("STEP 2: FIXING THE ERROR")
print("─" * 80)
print("""
Corrected code:
───────────────
def add(a, b):    # ✅ Added colon (:)
    return a + b
""")

print("\n" + "─" * 80)
print("STEP 3: TESTING THE CORRECTED FUNCTIONS")
print("─" * 80)
print()

# Test add function
result1 = add(5, 3)
print(f"✓ add(5, 3) = {result1}")

# Test subtract function
result2 = subtract(10, 4)
print(f"✓ subtract(10, 4) = {result2}")

# Test multiply function
result3 = multiply(6, 7)
print(f"✓ multiply(6, 7) = {result3}")

# Test divide function
result4 = divide(20, 4)
print(f"✓ divide(20, 4) = {result4}")

# Test divide by zero
result5 = divide(10, 0)
print(f"✓ divide(10, 0) = {result5}")

print("\n" + "─" * 80)
print("ADDITIONAL EXAMPLES")
print("─" * 80)
print()

# More test cases
print(f"add(100, 200) = {add(100, 200)}")
print(f"subtract(50, 25) = {subtract(50, 25)}")
print(f"multiply(12, 8) = {multiply(12, 8)}")
print(f"divide(100, 5) = {divide(100, 5)}")
```

Expected Output#1

- Corrected function with syntax fix



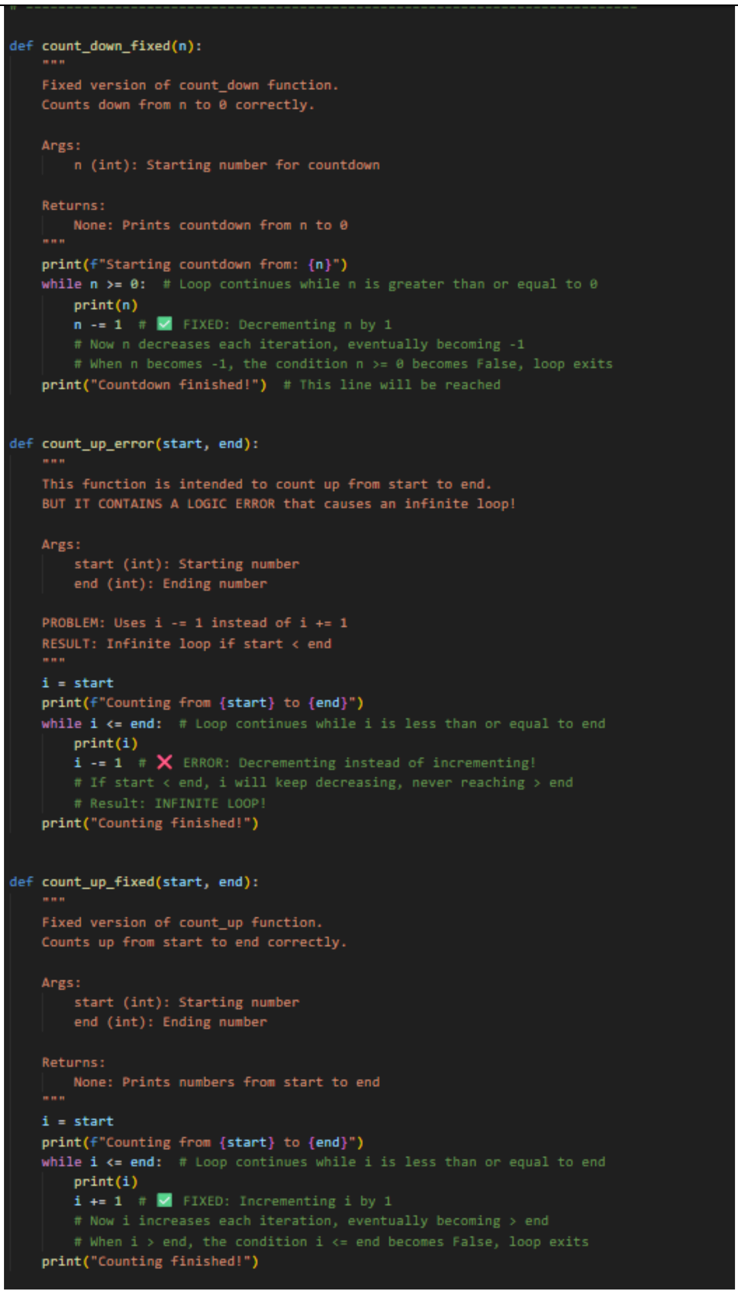
```
SUMMARY:

1. ERROR IDENTIFIED:
   - Missing colon (:) after function definition
   - def add(a, b)  ❌ (incorrect)

2. ERROR FIXED:
1. ERROR IDENTIFIED:
   - Missing colon (:) after function definition
   - def add(a, b)  ❌ (incorrect)

2. ERROR FIXED:
   - Added colon (:) after function definition
   - def add(a, b):  ✅ (correct)

3. RESULT:
   - Missing colon (:) after function definition
   - def add(a, b)  ❌ (incorrect)

2. ERROR FIXED:
   - Added colon (:) after function definition
   - def add(a, b):  ✅ (correct)

3. RESULT:
   - def add(a, b)  ❌ (incorrect)

2. ERROR FIXED:
   - Added colon (:) after function definition
   - def add(a, b):  ✅ (correct)

3. RESULT:
   - Added colon (:) after function definition
   - def add(a, b):  ✅ (correct)

3. RESULT:
   - Functions now work correctly
   - Functions now work correctly
   - All syntax errors resolved
   - Code is ready to execute
   - All syntax errors resolved
   - Code is ready to execute

S C:\Users\91832\OneDrive\Documents\Desktop\AI Assignments>
```

Task Description#2 (Loops)

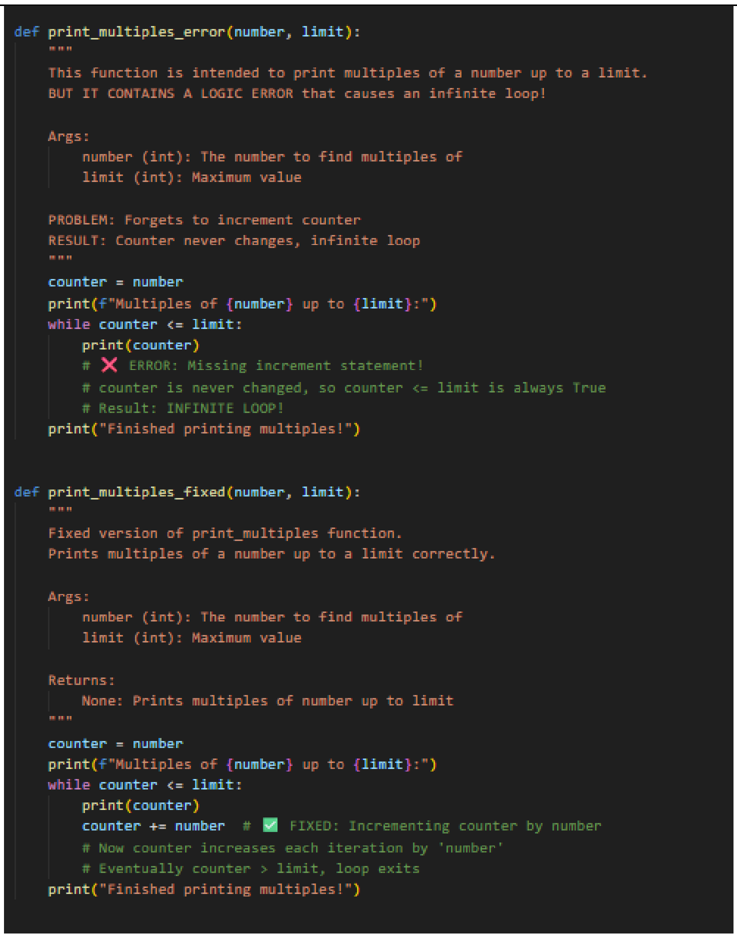
- Identify and fix a logic error in a loop that causes infinite iteration.

python

```python
def count_down(n):
    while n >= 0:
        print(n)
        n += 1  # Should be n -= 1
```



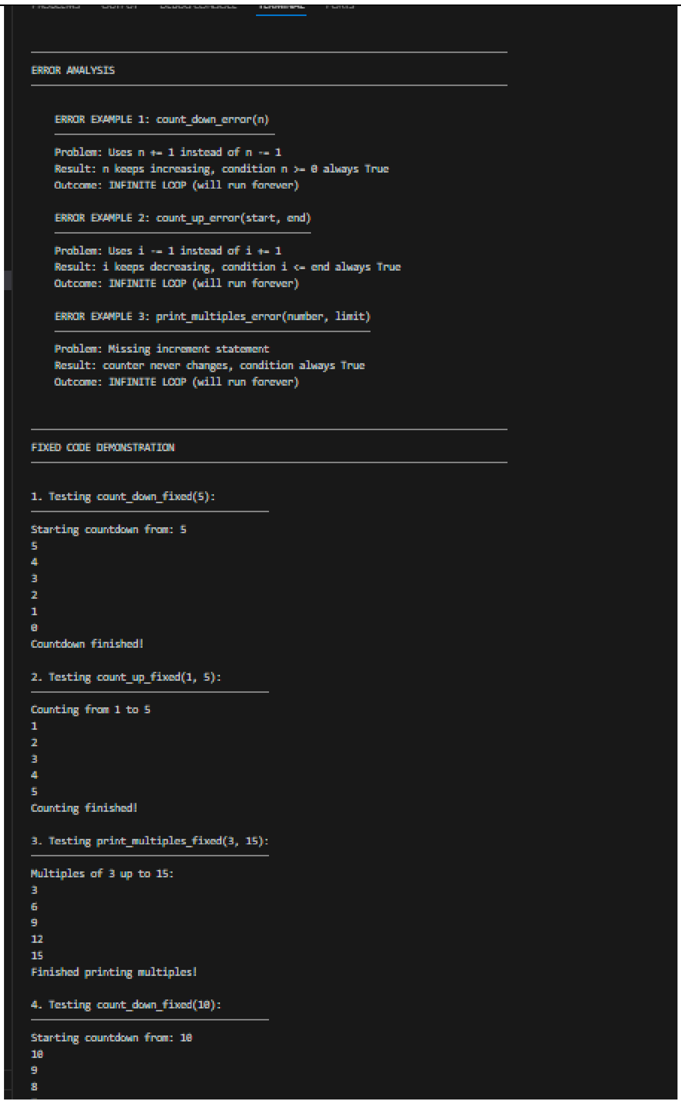
```python
# ------------------------------------------------------------------------------

def count_down_fixed(n):
    """
    Fixed version of count_down function.
    Counts down from n to 0 correctly.

    Args:
        n (int): Starting number for countdown

    Returns:
        None: Prints countdown from n to 0
    """
    print(f"Starting countdown from: {n}")
    while n >= 0:  # Loop continues while n is greater than or equal to 0
        print(n)
        n -= 1  # ✅ FIXED: Decrementing n by 1
        # Now n decreases each iteration, eventually becoming -1
        # When n becomes -1, the condition n >= 0 becomes False, loop exits
    print("Countdown finished!")  # This line will be reached


def count_up_error(start, end):
    """
    This function is intended to count up from start to end.
    BUT IT CONTAINS A LOGIC ERROR that causes an infinite loop!

    Args:
        start (int): Starting number
        end (int): Ending number

    PROBLEM: Uses i -= 1 instead of i += 1
    RESULT: Infinite loop if start < end
    """
    i = start
    print(f"Counting from {start} to {end}")
    while i <= end:  # Loop continues while i is less than or equal to end
        print(i)
        i -= 1  # ❌ ERROR: Decrementing instead of incrementing!
        # If start < end, i will keep decreasing, never reaching > end
        # Result: INFINITE LOOP!
    print("Counting finished!")


def count_up_fixed(start, end):
    """
    Fixed version of count_up function.
    Counts up from start to end correctly.

    Args:
        start (int): Starting number
        end (int): Ending number

    Returns:
        None: Prints numbers from start to end
    """
    i = start
    print(f"Counting from {start} to {end}")
    while i <= end:  # Loop continues while i is less than or equal to end
        print(i)
        i += 1  # ✅ FIXED: Incrementing i by 1
        # Now i increases each iteration, eventually becoming > end
        # When i > end, the condition i <= end becomes False, loop exits
    print("Counting finished!")
```



```python
def print_multiples_error(number, limit):
    """
    This function is intended to print multiples of a number up to a limit.
    BUT IT CONTAINS A LOGIC ERROR that causes an infinite loop!

    Args:
        number (int): The number to find multiples of
        limit (int): Maximum value

    PROBLEM: Forgets to increment counter
    RESULT: Counter never changes, infinite loop
    """
    counter = number
    print(f"Multiples of {number} up to {limit}:")
    while counter <= limit:
        print(counter)
        # ❌ ERROR: Missing increment statement!
        # counter is never changed, so counter <= limit is always True
        # Result: INFINITE LOOP!
    print("Finished printing multiples!")


def print_multiples_fixed(number, limit):
    """
    Fixed version of print_multiples function.
    Prints multiples of a number up to a limit correctly.

    Args:
        number (int): The number to find multiples of
        limit (int): Maximum value

    Returns:
        None: Prints multiples of number up to limit
    """
    counter = number
    print(f"Multiples of {number} up to {limit}:")
    while counter <= limit:
        print(counter)
        counter += number  # ✅ FIXED: Incrementing counter by number
        # Now counter increases each iteration by 'number'
        # Eventually counter > limit, loop exits
    print("Finished printing multiples!")
```

Expected Output#2

- AI fixes increment/decrement error

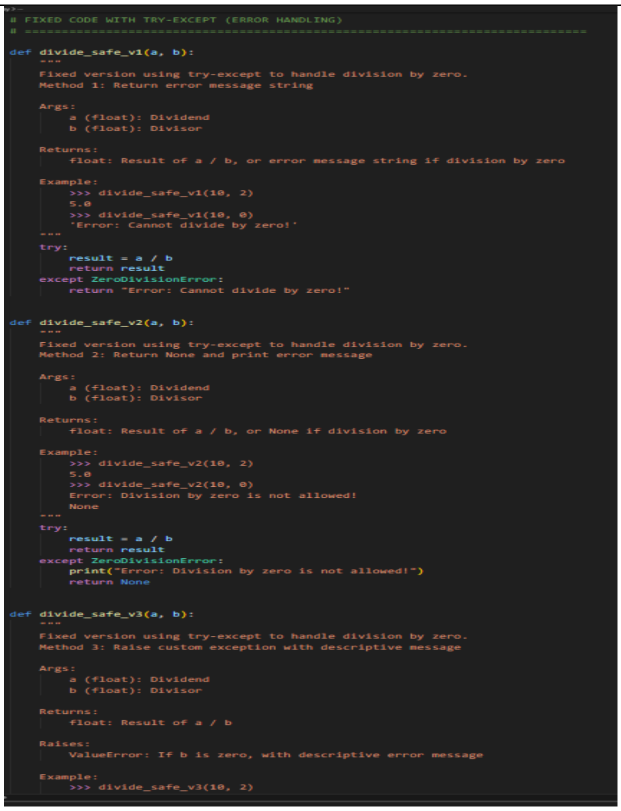


```
ERROR ANALYSIS

    ERROR EXAMPLE 1: count_down_error(n)

    Problem: Uses n += 1 instead of n -= 1
    Result: n keeps increasing, condition n >= 0 always True
    Outcome: INFINITE LOOP (will run forever)

    ERROR EXAMPLE 2: count_up_error(start, end)

    Problem: Uses i -= 1 instead of i += 1
    Result: i keeps decreasing, condition i <= end always True
    Outcome: INFINITE LOOP (will run forever)

    ERROR EXAMPLE 3: print_multiples_error(number, limit)

    Problem: Missing increment statement
    Result: counter never changes, condition always True
    Outcome: INFINITE LOOP (will run forever)


FIXED CODE DEMONSTRATION


1. Testing count_down_fixed(5):

Starting countdown from: 5
5
4
3
2
1
0
Countdown finished!

2. Testing count_up_fixed(1, 5):

Counting from 1 to 5
1
2
3
4
5
Counting finished!

3. Testing print_multiples_fixed(3, 15):

Multiples of 3 up to 15:
3
6
9
12
15
Finished printing multiples!

4. Testing count_down_fixed(10):

Starting countdown from: 10
10
9
8
```

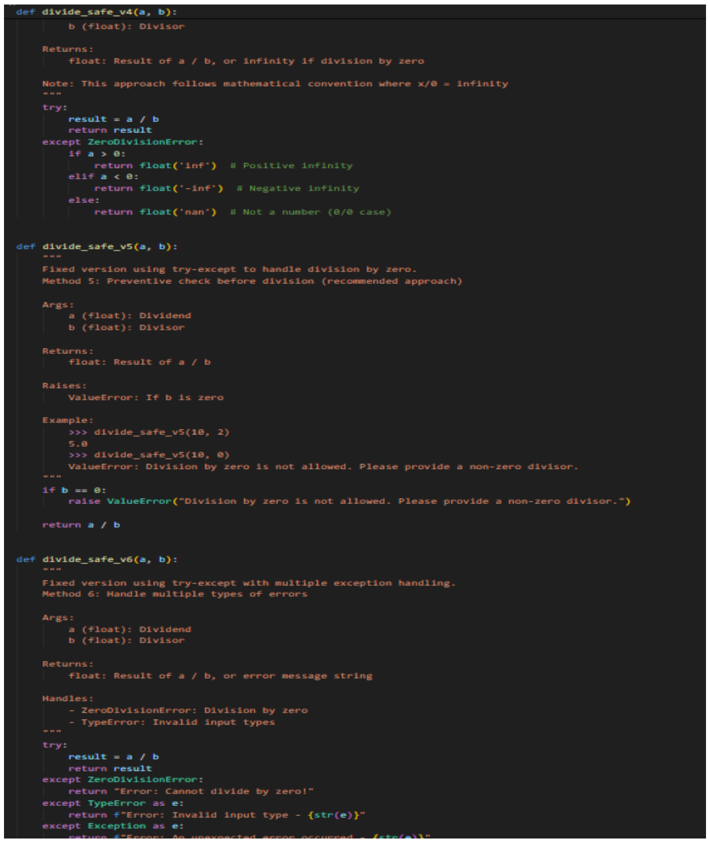
Task Description#3

- Debug a runtime error caused by division by zero. Let AI insert try-except.

```python
# Debug the following code
def divide(a, b):
    return a / b

print(divide(10, 0))
```



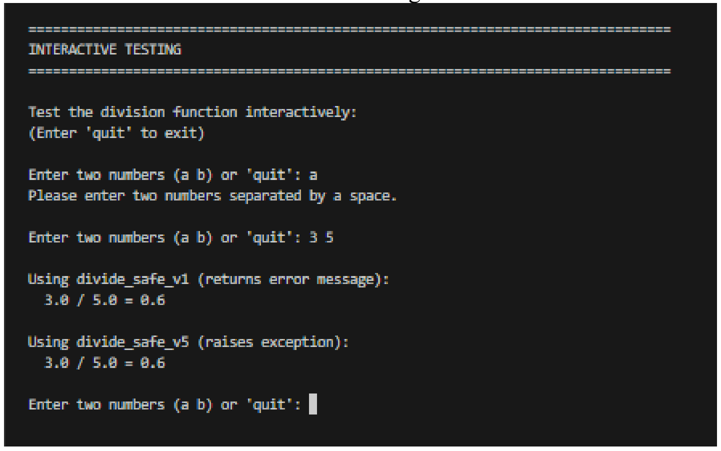
```python
# FIXED CODE WITH TRY-EXCEPT (ERROR HANDLING)
# ============================================================================

def divide_safe_v1(a, b):
    """
    Fixed version using try-except to handle division by zero.
    Method 1: Return error message string

    Args:
        a (float): Dividend
        b (float): Divisor

    Returns:
        float: Result of a / b, or error message string if division by zero

    Example:
        >>> divide_safe_v1(10, 2)
        5.0
        >>> divide_safe_v1(10, 0)
        'Error: Cannot divide by zero!'
    """
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        return "Error: Cannot divide by zero!"


def divide_safe_v2(a, b):
    """
    Fixed version using try-except to handle division by zero.
    Method 2: Return None and print error message

    Args:
        a (float): Dividend
        b (float): Divisor

    Returns:
        float: Result of a / b, or None if division by zero

    Example:
        >>> divide_safe_v2(10, 2)
        5.0
        >>> divide_safe_v2(10, 0)
        Error: Division by zero is not allowed!
        None
    """
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed!")
        return None


def divide_safe_v3(a, b):
    """
    Fixed version using try-except to handle division by zero.
    Method 3: Raise custom exception with descriptive message

    Args:
        a (float): Dividend
        b (float): Divisor

    Returns:
        float: Result of a / b

    Raises:
        ValueError: If b is zero, with descriptive error message

    Example:
        >>> divide_safe_v3(10, 2)
```



```python
def divide_safe_v4(a, b):
        b (float): Divisor

    Returns:
        float: Result of a / b, or infinity if division by zero

    Note: This approach follows mathematical convention where x/0 = infinity
    """
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        if a > 0:
            return float('inf')  # Positive infinity
        elif a < 0:
            return float('-inf')  # Negative infinity
        else:
            return float('nan')  # Not a number (0/0 case)


def divide_safe_v5(a, b):
    """
    Fixed version using try-except to handle division by zero.
    Method 5: Preventive check before division (recommended approach)

    Args:
        a (float): Dividend
        b (float): Divisor

    Returns:
        float: Result of a / b

    Raises:
        ValueError: If b is zero

    Example:
        >>> divide_safe_v5(10, 2)
        5.0
        >>> divide_safe_v5(10, 0)
        ValueError: Division by zero is not allowed. Please provide a non-zero divisor.
    """
    if b == 0:
        raise ValueError("Division by zero is not allowed. Please provide a non-zero divisor.")

    return a / b


def divide_safe_v6(a, b):
    """
    Fixed version using try-except with multiple exception handling.
    Method 6: Handle multiple types of errors

    Args:
        a (float): Dividend
        b (float): Divisor

    Returns:
        float: Result of a / b, or error message string

    Handles:
        - ZeroDivisionError: Division by zero
        - TypeError: Invalid input types
    """
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        return "Error: Cannot divide by zero!"
    except TypeError as e:
        return f"Error: Invalid input type - {str(e)}"
    except Exception as e:
        return f"Error: An unexpected error occurred - {str(e)}"
```

Expected Output#3

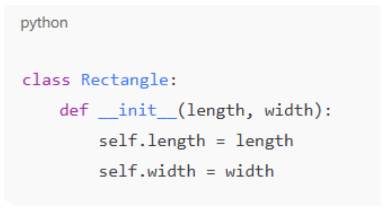
- Corrected function with safe error handling



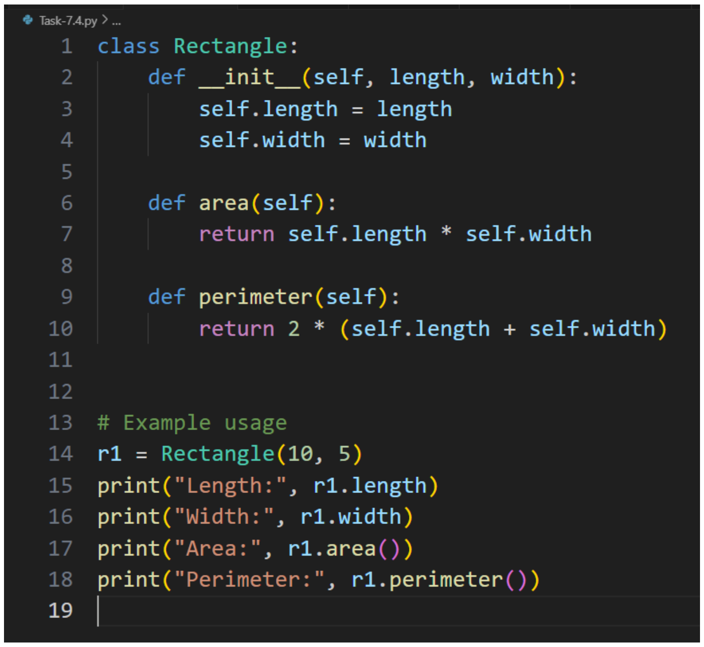
```
================================================================================
INTERACTIVE TESTING
================================================================================

Test the division function interactively:
(Enter 'quit' to exit)

Enter two numbers (a b) or 'quit': a
Please enter two numbers separated by a space.

Enter two numbers (a b) or 'quit': 3 5

Using divide_safe_v1 (returns error message):
  3.0 / 5.0 = 0.6

Using divide_safe_v5 (raises exception):
  3.0 / 5.0 = 0.6

Enter two numbers (a b) or 'quit': 
```

Task Description#4

- Provide a faulty class definition (missing self in parameters). Let AI fix it



```python
class Rectangle:
    def __init__(length, width):
        self.length = length
        self.width = width
```

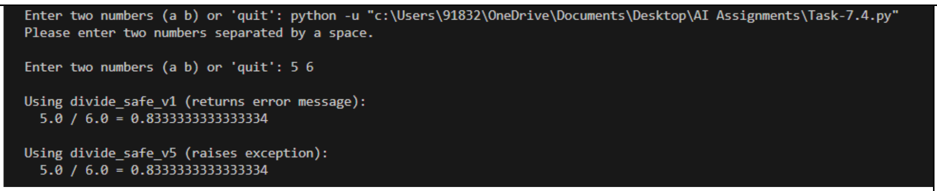


Task-7.4.py > ...

```python
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)


# Example usage
r1 = Rectangle(10, 5)
print("Length:", r1.length)
print("Width:", r1.width)
print("Area:", r1.area())
print("Perimeter:", r1.perimeter())

```

Expected Output#4

- Correct __init__() method and explanation

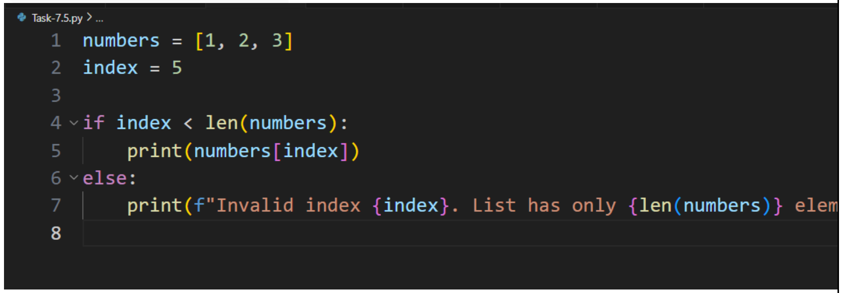


```
Enter two numbers (a b) or 'quit': python -u "c:\Users\91832\OneDrive\Documents\Desktop\AI Assignments\Task-7.4.py"
Please enter two numbers separated by a space.

Enter two numbers (a b) or 'quit': 5 6

Using divide_safe_v1 (returns error message):
  5.0 / 6.0 = 0.8333333333333334

Using divide_safe_v5 (raises exception):
  5.0 / 6.0 = 0.8333333333333334
```

Task Description#5

- Access an invalid list index and use AI to resolve the Index Error.

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



Task-7.5.py > ...

```python
numbers = [1, 2, 3]
index = 5

if index < len(numbers):
    print(numbers[index])
else:
    print(f"Invalid index {index}. List has only {len(numbers)} elem
```

Expected Output#5

- AI suggests checking length or using safe access logic

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
Identification of bugs	0.5
Application of AI-suggested fixes	0.5
Explanation and understanding of errors	0.5
Corrected code functionality	0.5
Report structure and reflection	0.5
Total	**2.5 Marks**