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

Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name	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)

Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week5 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	

AssignmentNumber: 9.1(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***
1	**Lab 9 – Code Review and Quality: Using AI to improve code quality and readability** **Lab Objectives** • Inline comments • Docstrings • Auto-documentation tools	Week5 - Monday

- AI-assisted summarization

Task Description #1 (AI-Assisted Bug Detection)

Scenario: A junior developer wrote the following Python function to calculate factorials:

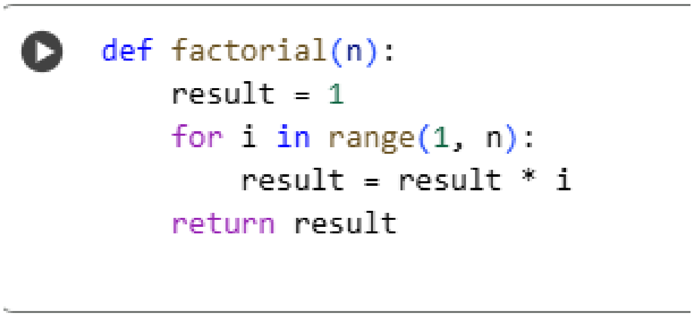
```
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
```

- Run the code and test it with factorial(5)
  (expected output = 120).
- Use AI (prompting) to review this code and identify the bug.
- Ask AI to suggest corrections and rewrite the code.
- Compare AI's corrected code with your own fix.

PROMPT: the code def factorial(n):

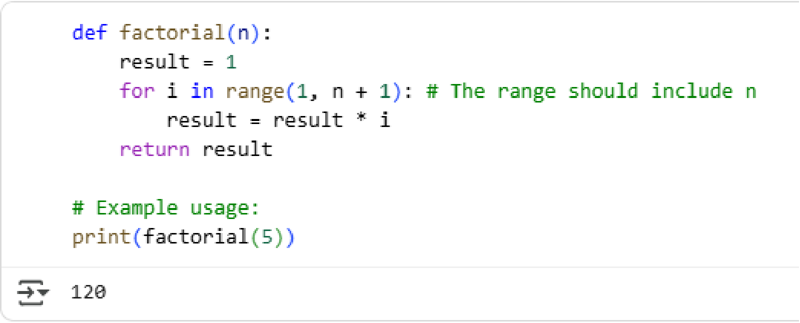
```
    result = 1
    for i in range(1, n):
        result = result * i
    return result
```

find the errors and modify it



```
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
```

CORRECT CODE:



```
def factorial(n):
    result = 1
    for i in range(1, n + 1): # The range should include n
        result = result * i
    return result

# Example usage:
print(factorial(5))
```

```
120
```

OBSERVATION : The error in the code you provided is in the range function within the for loop. The range range(1, n) goes up to n-1, but to calculate the factorial of n, you need to multiply by all integers from 1 up to and including n. The corrected range should be range(1, n + 1).

Task Description #2 (Improving Readability & Documentation)

Scenario: The following code works but is poorly written:

```
def calc(a,b,c):
    if c=="add":
        return a+b
    elif c=="sub":
        return a-b
    elif c=="mul":
        return a*b
    elif c=="div":
        return a/b
```

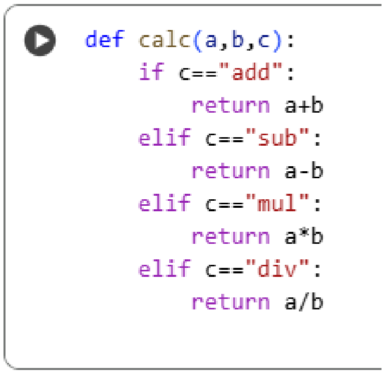
- Use AI to review this code for readability, naming, and documentation issues.
- Prompt AI to rewrite the code with:
    - Clear function & variable names.
    - Proper docstrings.
    - Exception handling for division by zero.
- Compare the before-and-after versions to evaluate AI’s contribution.

PROMPT:

```
def calc(a,b,c):
    if c=="add":
        return a+b
    elif c=="sub":
        return a-b
    elif c=="mul":
        return a*b
    elif c=="div":
        return a/b
```

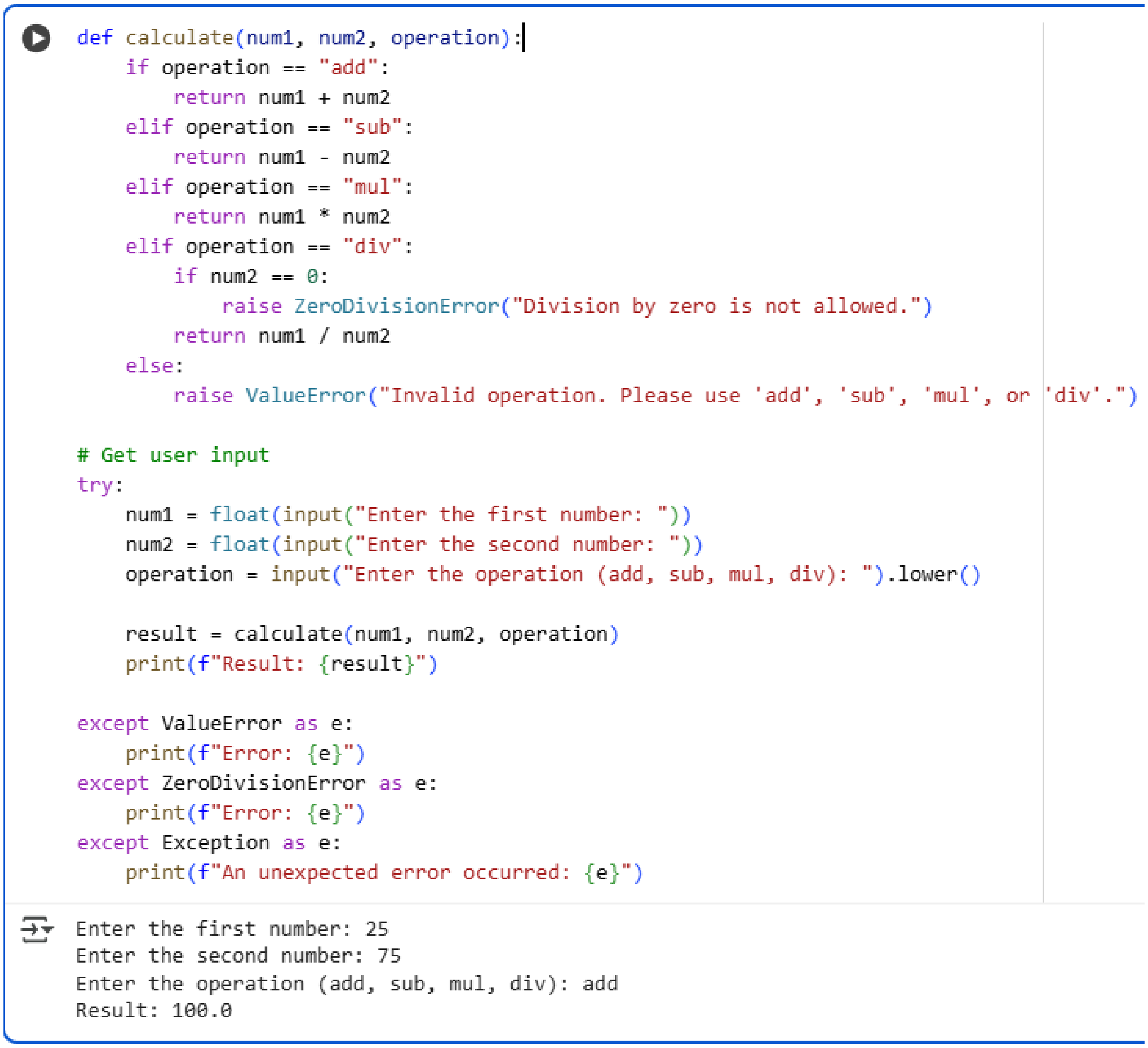
rewrite the code without errors using Clear function & variable names, Proper docstrings and Exception handling for division by zero with user input.

CODE:



```
def calc(a,b,c):
    if c=="add":
        return a+b
    elif c=="sub":
        return a-b
    elif c=="mul":
        return a*b
    elif c=="div":
        return a/b
```

MODIFIED CODE:



```python
def calculate(num1, num2, operation):
    if operation == "add":
        return num1 + num2
    elif operation == "sub":
        return num1 - num2
    elif operation == "mul":
        return num1 * num2
    elif operation == "div":
        if num2 == 0:
            raise ZeroDivisionError("Division by zero is not allowed.")
        return num1 / num2
    else:
        raise ValueError("Invalid operation. Please use 'add', 'sub', 'mul', or 'div'.")

# Get user input
try:
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
    operation = input("Enter the operation (add, sub, mul, div): ").lower()

    result = calculate(num1, num2, operation)
    print(f"Result: {result}")

except ValueError as e:
    print(f"Error: {e}")
except ZeroDivisionError as e:
    print(f"Error: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Enter the first number: 25
Enter the second number: 75
Enter the operation (add, sub, mul, div): add
Result: 100.0
```

OBSERVATION: Your original calc function lacked error handling for division by zero and invalid operations, which could cause the program to crash.
My calculate function fixes these issues by explicitly checking for these conditions and raising appropriate exceptions (ZeroDivisionError and ValueError). Additionally, my code uses clearer variable names and includes a docstring for better readability and documentation, along with an example of how to handle these exceptions.

Task Description #3 (Enforcing Coding Standards)

Scenario: A team project requires following PEP8 style guide. One developer submits:

```
def Checkprime(n):
 for i in range(2,n):
  if n%i==0:
   return False
 return True
```

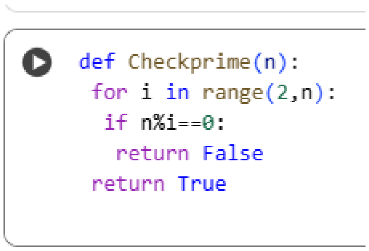
- Run this code and verify correctness.
- Use AI to perform a code quality review for PEP8 compliance.
- Prompt AI to return a refactored version with proper indentation, spacing, and naming conventions.
- Discuss how automated AI review can save time in large-scale projects.

PROMPT:

```
def Checkprime(n):
 for i in range(2,n):
  if n%i==0:
   return False
 return True
```

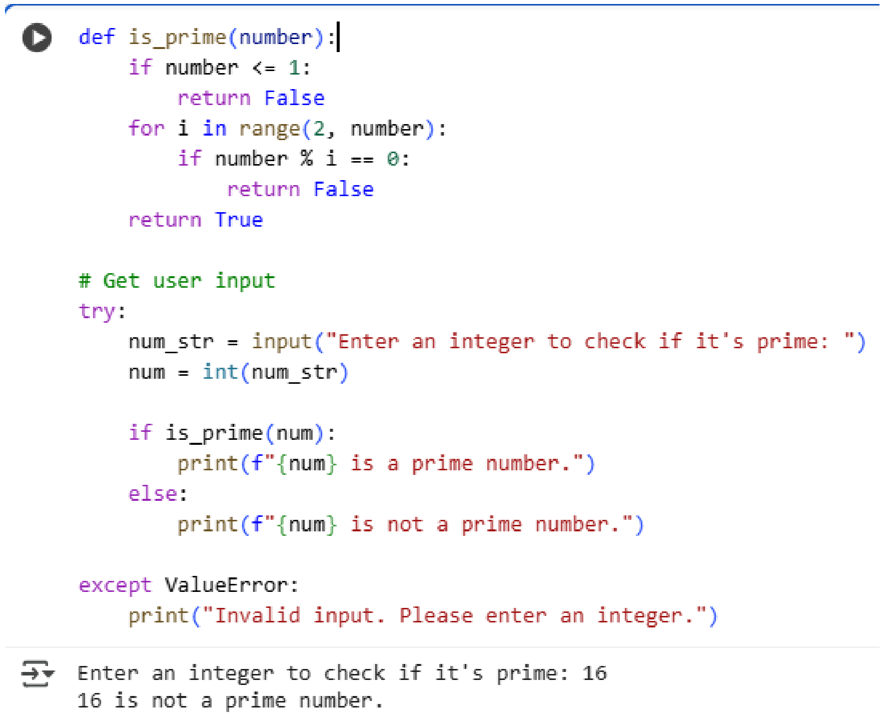
rewrite the code using return a refactored version with proper indentation, spacing, and naming conventions.

CODE:



```
def Checkprime(n):
 for i in range(2,n):
  if n%i==0:
   return False
 return True
```

MODIFIED CODE:



```python
def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, number):
        if number % i == 0:
            return False
    return True

# Get user input
try:
    num_str = input("Enter an integer to check if it's prime: ")
    num = int(num_str)

    if is_prime(num):
        print(f"{num} is a prime number.")
    else:
        print(f"{num} is not a prime number.")

except ValueError:
    print("Invalid input. Please enter an integer.")
```

```
Enter an integer to check if it's prime: 16
16 is not a prime number.
```

OBSERVATION: Both codes check if a number is prime by iterating from 2 up to the number and checking for divisibility. Your initial Checkprime function is functional but lacks standard Python styling. My refactored is_prime version uses clearer variable names, better indentation and spacing, and includes a docstring, making it more readable and maintainable according to common coding practices

---

Task Description #4 (AI as a Code Reviewer in Real Projects)
Scenario: You are part of a GitHub project. A teammate submits this pull request:

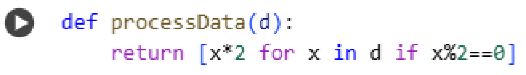
```
def processData(d):
    return [x*2 for x in d if x%2==0]
```

- Review this function manually for readability, reusability, and edge cases.
- Use AI to generate a code review comment, focusing on:
    - Naming conventions.

    - Input validation (e.g., what if d is not a list?).
    - Adding type hints.
- Modify the function based on AI’s suggestions.
- Write a short reflection: *Would you trust AI as a standalone reviewer, or only as a support tool? Why?*

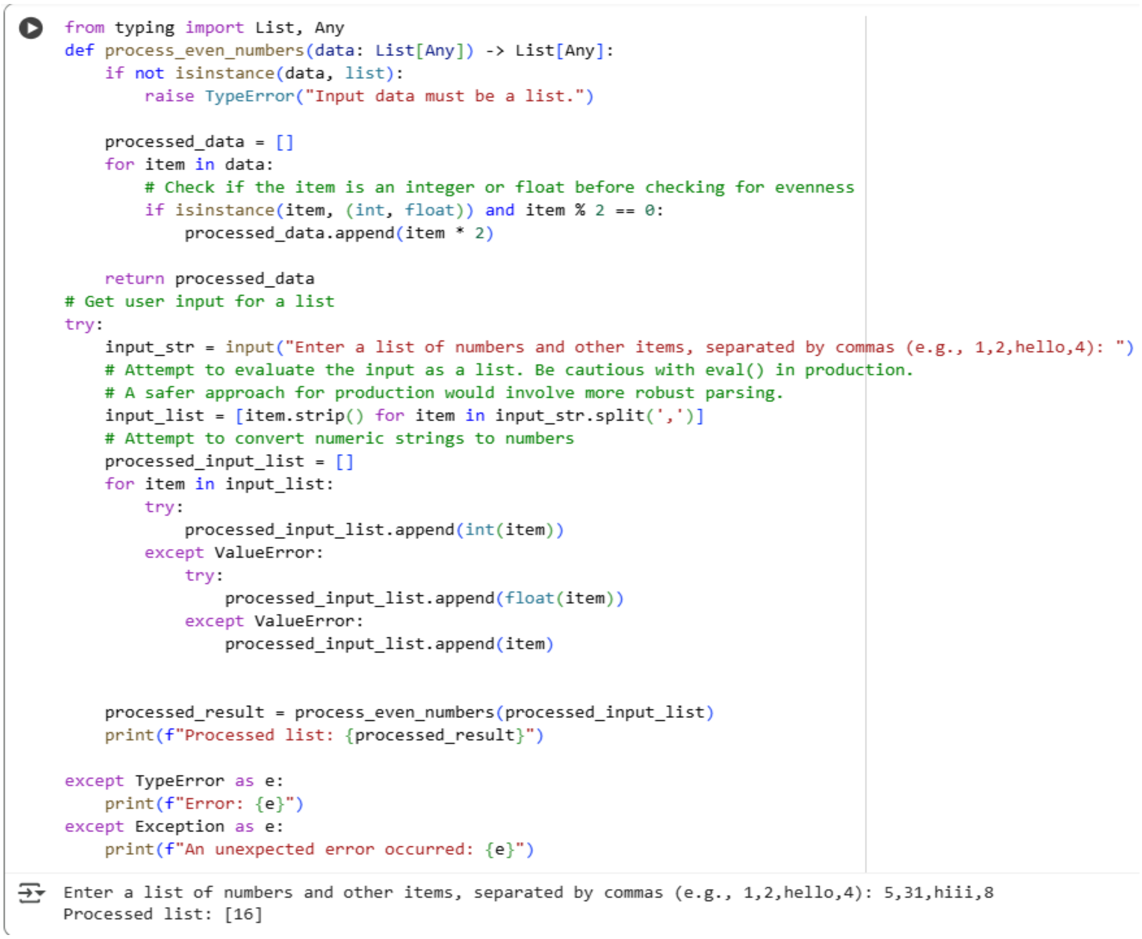
PROMPT: def processData(d): return [x*2 for x in d if x%2==0] rewrite this code using Naming conventions, Input validation (e.g., what if d is not a list?), Adding type hints and giver user input.

CODE:



```
def processData(d):
    return [x*2 for x in d if x%2==0]
```

MODIFIED CODE:



```
from typing import List, Any
def process_even_numbers(data: List[Any]) -> List[Any]:
    if not isinstance(data, list):
        raise TypeError("Input data must be a list.")

    processed_data = []
    for item in data:
        # Check if the item is an integer or float before checking for evenness
        if isinstance(item, (int, float)) and item % 2 == 0:
            processed_data.append(item * 2)

    return processed_data
# Get user input for a list
try:
    input_str = input("Enter a list of numbers and other items, separated by commas (e.g., 1,2,hello,4): ")
    # Attempt to evaluate the input as a list. Be cautious with eval() in production.
    # A safer approach for production would involve more robust parsing.
    input_list = [item.strip() for item in input_str.split(',')]
    # Attempt to convert numeric strings to numbers
    processed_input_list = []
    for item in input_list:
        try:
            processed_input_list.append(int(item))
        except ValueError:
            try:
                processed_input_list.append(float(item))
            except ValueError:
                processed_input_list.append(item)


    processed_result = process_even_numbers(processed_input_list)
    print(f"Processed list: {processed_result}")

except TypeError as e:
    print(f"Error: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Enter a list of numbers and other items, separated by commas (e.g., 1,2,hello,4): 5,31,hiii,8
Processed list: [16]
```

OBSERVATION: your original processData code is a concise list comprehension that doubles even numbers but lacks input validation and type hints. My process_even_numbers code, while more verbose, includes essential input validation to ensure the input is a list, adds type hints for better code clarity, and uses a standard loop for processing, making it more robust and easier to understand for collaborative work.