Lab Assisgnment- 10.2

Program : B. Tech (CSE)

Specialization : AIML

Course Title : AI Assisted coding

Semester : III

Academic Session : 2025-2026

Name of Student : Bommineni Spandana

Enrollment No : 2403a52005

Batch No. : 01

Date : 07-10-2025

Task Description#1 Assisted

Code Review (Basic Errors)

• Write python program as shown below.

• Use an AI assistant to review and suggest corrections

Expected Outcome#1: Students need to submit corrected code with comments

Prompt :

Here's a Python program written to calculate the factorial of a number. The loop uses the variable i, but the goal is to rewrite or correct the code without using i as a loop variable. Also, please check for any basic errors and suggest improvements with comments.

```python
def calcFact(n):
    result=1
    x=0
    for i in range(1,n):
        result=result*i
    return result


def main():
    num = 5
    FACT = calcFact(num)
    print("the factorial of",num,"is",FACT)
    t=10
    if FACT>10:
            print("BIG Number")
    else:
      print("small number")

main()
```

Corrected code:

```python
def calcFact(n):
    # Initialize the result to 1. The factorial of 0 is 1.
    result = 1
    # The variable 'x' is not used in the original code and can be removed.
    # x = 0

    # Iterate through the numbers from 1 up to (but not including) n.
    # The original code calculates factorial of n-1.
    # To calculate factorial of n, the range should be from 1 to n+1.
    # Also, the loop variable name is changed from 'i' to 'num'.
    for num in range(1, n + 1):
        result = result * num
    return result

def main():
    # Get user input for the number and convert it to an integer.
    try:
        num_str = input("Enter a non-negative integer to calculate its factorial: ")
        num = int(num_str)

        # Check if the input is a non-negative integer.
        if num < 0:
            print("Factorial is not defined for negative numbers.")
            return
```

```
        if num < 0:
            print("Factorial is not defined for negative numbers.")
            return

        # It's good practice to use lowercase for variable names, FACT is renamed to fact_result.
        fact_result = calcFact(num)
        print("the factorial of", num, "is", fact_result)

        # Check if the calculated factorial is greater than 10.
        if fact_result > 10:
            print("BIG Number")
        else:
            print("small number")
    except ValueError:
        print("Invalid input. Please enter an integer.")


# Call the main function to execute the program.
main()
```

```
Enter a non-negative integer to calculate its factorial: 5
the factorial of 5 is 120
BIG Number
```

Task Description#2 Automatic Inline Comments

• Write the Python code for Fibonacci as shown below and execute.

• Ask AI to improve variable names, add comments, and apply PEP8 formatting

(cleaned up).

• Students evaluate which suggestions improve readability most. One

Expected Output#2: Clean format python code with much readability.

Prompt:

A Python program has been written to calculate the sum of Fibonacci numbers up to a given count. The task involves improving the code by:- Replacing unclear variable names- Adding meaningful comments- Applying PEP8 formatting The loop currently uses the variable i, which should be avoided. review the code and suggest a corrected version with better readability:

```python
def f1(xX):
    a=0
    b=1
    c=2
    Zz=[a,b]
    while c<=xX:
        d=a+b
        Zz.append(d)
        a=b
        b=d
        c=c+1
    return Zz

def m():
    NN=10
    ans=f1(NN)
    print("fib series till",NN,":",ans)

m()
```

Corrected code:

```python
def calculate_fibonacci_sum(count):
    """
    Calculates the sum of Fibonacci numbers up to a given count.

    Args:
        count: The number of Fibonacci numbers to sum.

    Returns:
        The sum of the first 'count' Fibonacci numbers.
    """
    # Initialize the first two Fibonacci numbers and the sum.
    fib1 = 0
    fib2 = 1
    current_count = 2  # Start from the third number since the first two are initialized.
    fib_sum = 1 # Initialize sum with fib2 as fib1 is 0

    # Loop to generate Fibonacci numbers and calculate their sum.
    # The loop continues as long as the current count is less than or equal to the desired count.
    while current_count <= count:
        next_fib = fib1 + fib2
        fib_sum = fib_sum + next_fib
        # Update the previous two Fibonacci numbers for the next iteration.
        fib1 = fib2
        fib2 = next_fib
        # Increment the count of generated Fibonacci numbers.
```

```python
        # Increment the count of generated Fibonacci numbers.
        current_count = current_count + 1

    return fib_sum

def main():
    # Get user input for the number of Fibonacci numbers to sum.
    try:
        input_count_str = input("Enter the number of Fibonacci numbers to sum: ")
        input_count = int(input_count_str)

        # Check if the input is a non-negative integer.
        if input_count < 0:
            print("Count cannot be negative.")
            return
        elif input_count == 0:
            print("The sum of the first 0 Fibonacci numbers is 0.")
            return
        elif input_count == 1:
            print("The sum of the first 1 Fibonacci numbers is 0.")
            return


        # Calculate the sum of Fibonacci numbers using the improved function.
        result_sum = calculate_fibonacci_sum(input_count)
```

```python
        print("The sum of the first 1 Fibonacci numbers is 0.")
        return


        # Calculate the sum of Fibonacci numbers using the improved function.
        result_sum = calculate_fibonacci_sum(input_count)
        print("The sum of the first", input_count, "Fibonacci numbers is", result_sum)

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

# Call the main function to execute the program.
main()
```

```
Enter the number of Fibonacci numbers to sum: 22
The sum of the first 22 Fibonacci numbers is 46367
```

Task Description#3

• Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).

• Incorporate manual docstring in code with NumPy Style

• Use AI assistance to generate a module-level docstring + individual function docstrings.

• Compare the AI-generated docstring with your manually written one

Common Examples of Code Smells

• Long Function – A single function tries to do too many things.

• Duplicate Code – Copy-pasted logic in multiple places.

• Poor Naming – Variables or functions with confusing names (x1, foo, data123).

• Unused Variables – Declaring variables but never using them.

• Magic Numbers – Using unexplained constants (3.14159 instead of PI).

• Deep Nesting – Too many if/else levels, making code hard to read.

• Large Class – A single class handling too many responsibilities.

Why Detecting Code Smells is Important

• Makes code easier to read and maintain.

• Reduces chance of bugs in future updates.

• Helps in refactoring (improving structure without changing behavior)

• Encourages clean coding practices

Dead Code – Code that is never executed.

Expected Output#3: Students learn structured documentation for multi-function scripts

Prompt:

Create a python program with four functions: add, subtract, multiply, and divide. Each function includes a manually written docstring using NumPy style.

 Assistance is requested to:

- Generate a module-level docstring
- Rewrite each function's docstring using NumPy style
- Identify any code smells such as long functions, duplicate logic, poor naming, unused variables, magic numbers, deep nesting, or dead code
- Suggest improvements for readability and structure

 A comparison will be made between manually written docstrings and those generated by the assistant to evaluate clarity and effectiveness.

Code:

First function: Addition

```python
def add(a, b):
    """
    Adds two numbers.

    Parameters
    ----------
    a : int or float
        The first number.
    b : int or float
        The second number.

    Returns
    -------
    int or float
        The sum of a and b.
    """
    return a + b
```

Second Function: Subtraction

```python
def subtract(a, b):
    """
    Subtracts the second number from the first.

    Parameters
    ----------
    a : int or float
        The first number.
    b : int or float
        The second number.

    Returns
    -------
    int or float
        The difference between a and b.
    """
    return a - b
```

Third function: Multiplication

```python
def multiply(a, b):
    """
    Multiplies two numbers.

    Parameters
    ----------
    a : int or float
        The first number.
    b : int or float
        The second number.

    Returns
    -------
    int or float
        The product of a and b.
    """
    return a * b
```

Fourth function: Division

```python
def divide(a, b):
    """
    Divides the first number by the second, handling division by zero.

    Parameters
    ----------
    a : int or float
        The numerator.
    b : int or float
        The denominator.

    Returns
    -------
    int or float or str
        The quotient of a and b, or an error message string if division by zero occurs.
    """
    # Code smell: Using a magic string "Error: Division by zero is not allowed."
    # Improvement: Define a constant for the error message.
    DIVISION_BY_ZERO_ERROR = "Error: Division by zero is not allowed."
    if b == 0:
        return DIVISION_BY_ZERO_ERROR
    return a / b
```

```python
def main():
    """
    Main function to handle user input and perform arithmetic operations.
    """
    try:
        # Code smell: Repetitive input and conversion logic for num1 and num2.
        # Improvement: Create a helper function to get validated numerical input.
        num1_str = input("Enter the first number: ")
        num1 = float(num1_str)

        num2_str = input("Enter the second number: ")
        num2 = float(num2_str)

        operation = input("Enter the operation (+, -, *, /): ")

        # Code smell: Long if-elif-else block for operation handling.
        # Improvement: Use a dictionary to map operations to functions.
        if operation == '+':
            result = add(num1, num2)
        elif operation == '-':
            result = subtract(num1, num2)
        elif operation == '*':
            result = multiply(num1, num2)
        elif operation == '/':
            result = divide(num1, num2)
```
```python
        result = multiply(num1, num2)
        elif operation == '/':
            result = divide(num1, num2)
        else:
            # Code smell: Using a magic string for invalid operation.
            # Improvement: Define a constant for the invalid operation message.
            INVALID_OPERATION_ERROR = "Invalid operation. Please enter one of +, -, *, /."
            print(INVALID_OPERATION_ERROR)
            return

        print(f"The result is: {result}")

    except ValueError:
        # Code smell: Using a magic string for invalid input.
        # Improvement: Define a constant for the invalid input message.
        INVALID_INPUT_ERROR = "Invalid input. Please enter valid numbers."
        print(INVALID_INPUT_ERROR)

# Code smell: Lack of a clear function to handle input validation and conversion.
# Improvement: Create a function like `get_number_input` to handle this.
# Code smell: Repetitive error message strings throughout the code.
# Improvement: Define constants for common error messages.

if __name__ == "__main__":
    main()
```

Output:

```
Enter the first number: 10
Enter the second number: 22
Enter the operation (+, -, *, /): *
The result is: 220.0
```