AI ASSISTED CODING

ASSIGNMENT-10.4

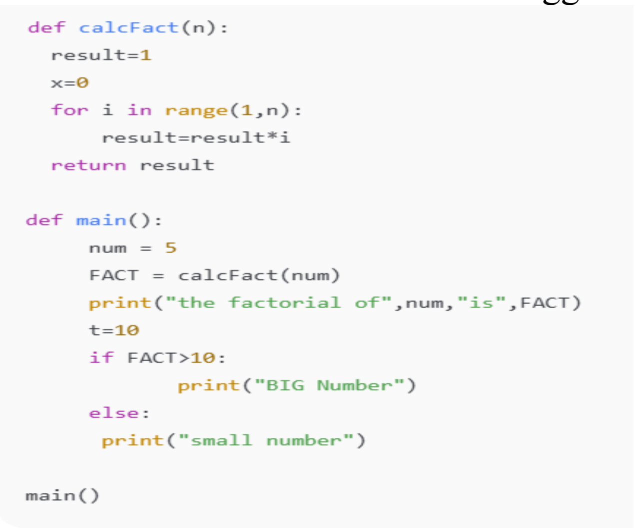
NAME:SIMRA TAHSEEN

ROLL NO:2503A51L017

BATCH: 24BTCAICSB19

Task Description#1 AI-Assisted Code Review (Basic Errors)

- Write python program as shown below.
- Use an AI assistant to review and suggest corrections



```
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()
```

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

PROMPT: Review the above code and suggest corrections.
rewrite the correct code with comments.

Corrections made in the code:

✓ The loop for i in range(1, n): is incorrect. It should be range(1, n + 1) to include n itself.

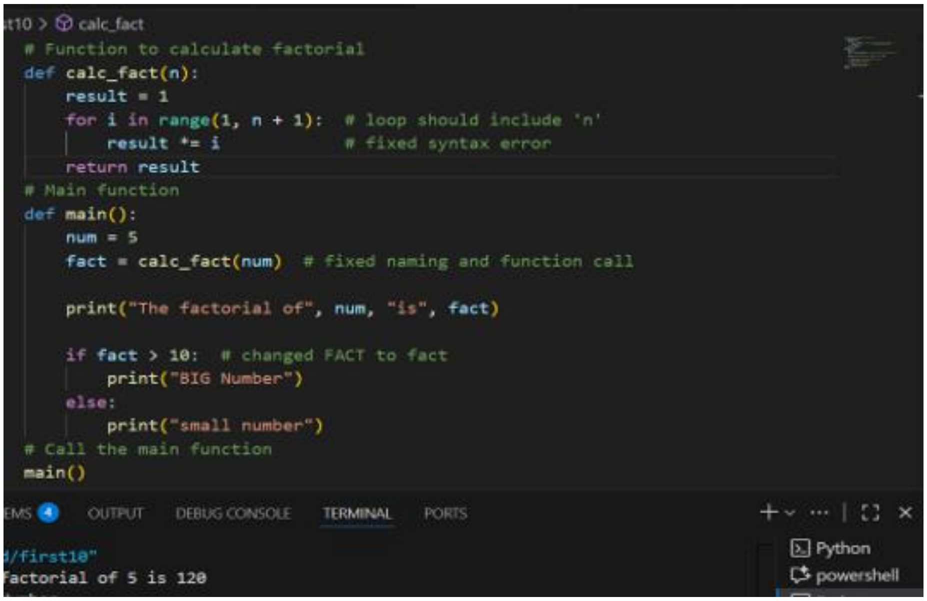
✓ Variable x = 0 in calcFact() is never used.

✓ In main(), num is defined, but passed to calcFact() as num, yet the function is called before num is declared as a parameter.

✓ The variable t = 10 is declared but unused.

✓ The if FACT > 10: is valid, but t being declared above has no effect.

- ✓ Use proper naming conventions (e.g., `fact` instead of `FACT`).
- ✓ Ensure consistent indentation and spacing for readability.

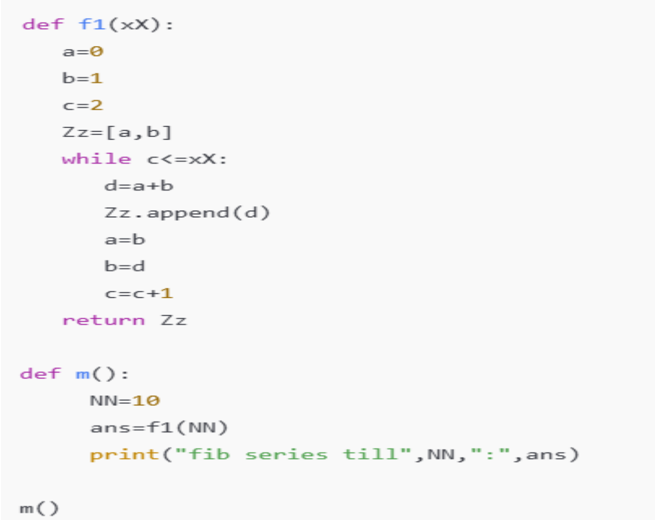


```python
# Function to calculate factorial
def calc_fact(n):
    result = 1
    for i in range(1, n + 1):  # loop should include 'n'
        result *= i            # fixed syntax error
    return result
# Main function
def main():
    num = 5
    fact = calc_fact(num)  # fixed naming and function call

    print("The factorial of", num, "is", fact)

    if fact > 10:  # changed FACT to fact
        print("BIG Number")
    else:
        print("small number")
# Call the main function
main()
```

```
d/first10"
actorial of 5 is 120
```

Task Description#2 Automatic Inline Comments

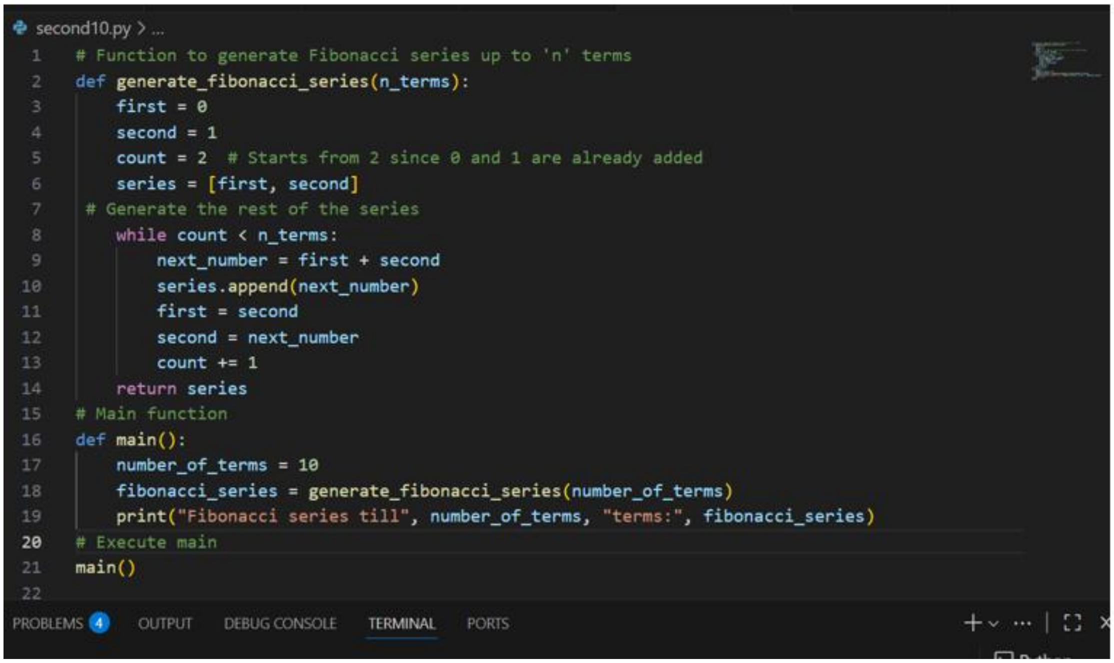
- Write the Python code for Fibonacci as shown below and execute.
- Ask AI to improve variable names, add comments, and apply PEP8 formatting (cleaned up).
- Students evaluate which suggestions improve readability most. one.



```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()
```

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

PROMPT: improve variable names, add comments, and apply PEP8 formatting (cleaned up) for the above python code



```python
# Function to generate Fibonacci series up to 'n' terms
def generate_fibonacci_series(n_terms):
    first = 0
    second = 1
    count = 2  # Starts from 2 since 0 and 1 are already added
    series = [first, second]
 # Generate the rest of the series
    while count < n_terms:
        next_number = first + second
        series.append(next_number)
        first = second
        second = next_number
        count += 1
    return series
# Main function
def main():
    number_of_terms = 10
    fibonacci_series = generate_fibonacci_series(number_of_terms)
    print("Fibonacci series till", number_of_terms, "terms:", fibonacci_series)
# Execute main
main()
```

Key Improvements:

- Replacing `a, b, c, Zz` with `first, second, count, series` makes the logic self-explanatory.
- Function name `generate_fibonacci_series` tells you exactly what the function does.

- Original used `while c <= xX`, which actually **generates `xX + 1` terms**, but your corrected code uses `while count < n_terms`, which properly limits the list to **exactly `n_terms` values**.

- Each logical block now has a comment — helps learners or future readers quickly understand what the code is doing.

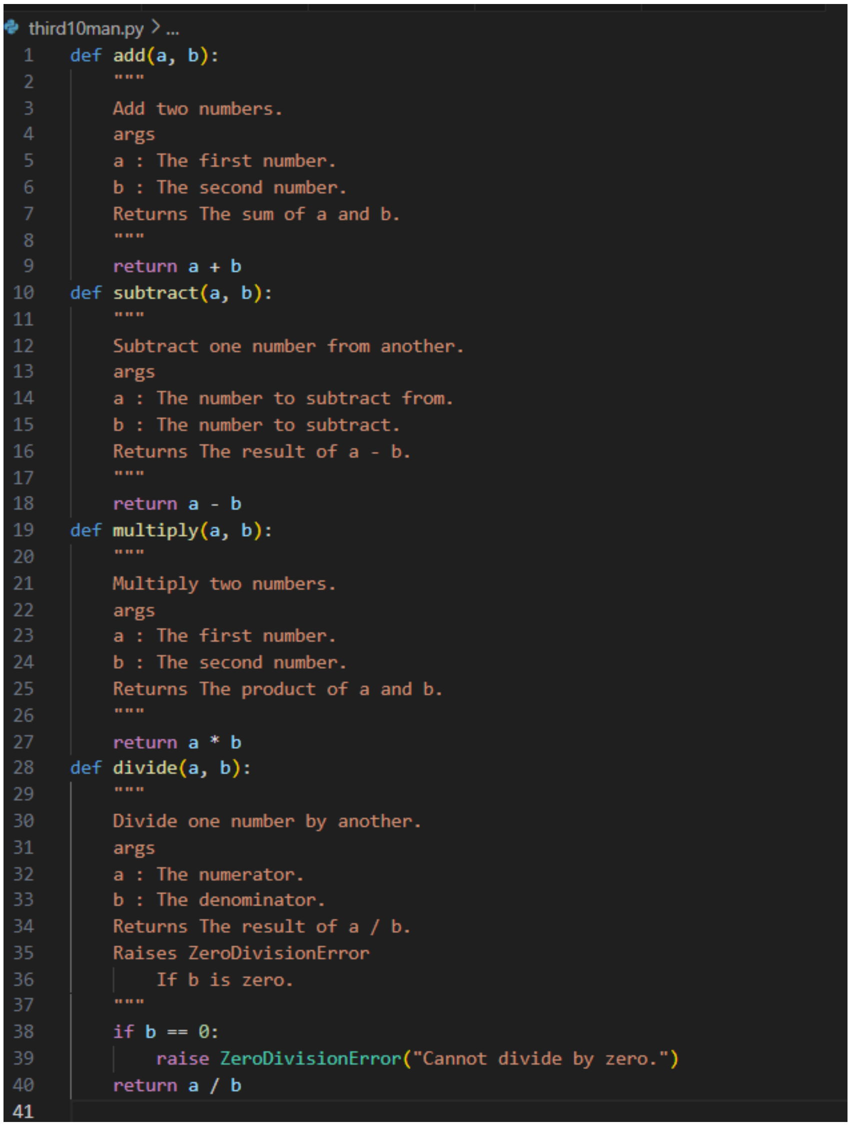
- PEP8 style: consistent spacing, indentation, and line length.
- Your code looks professional and beginner-friendly now.

Task Description#3

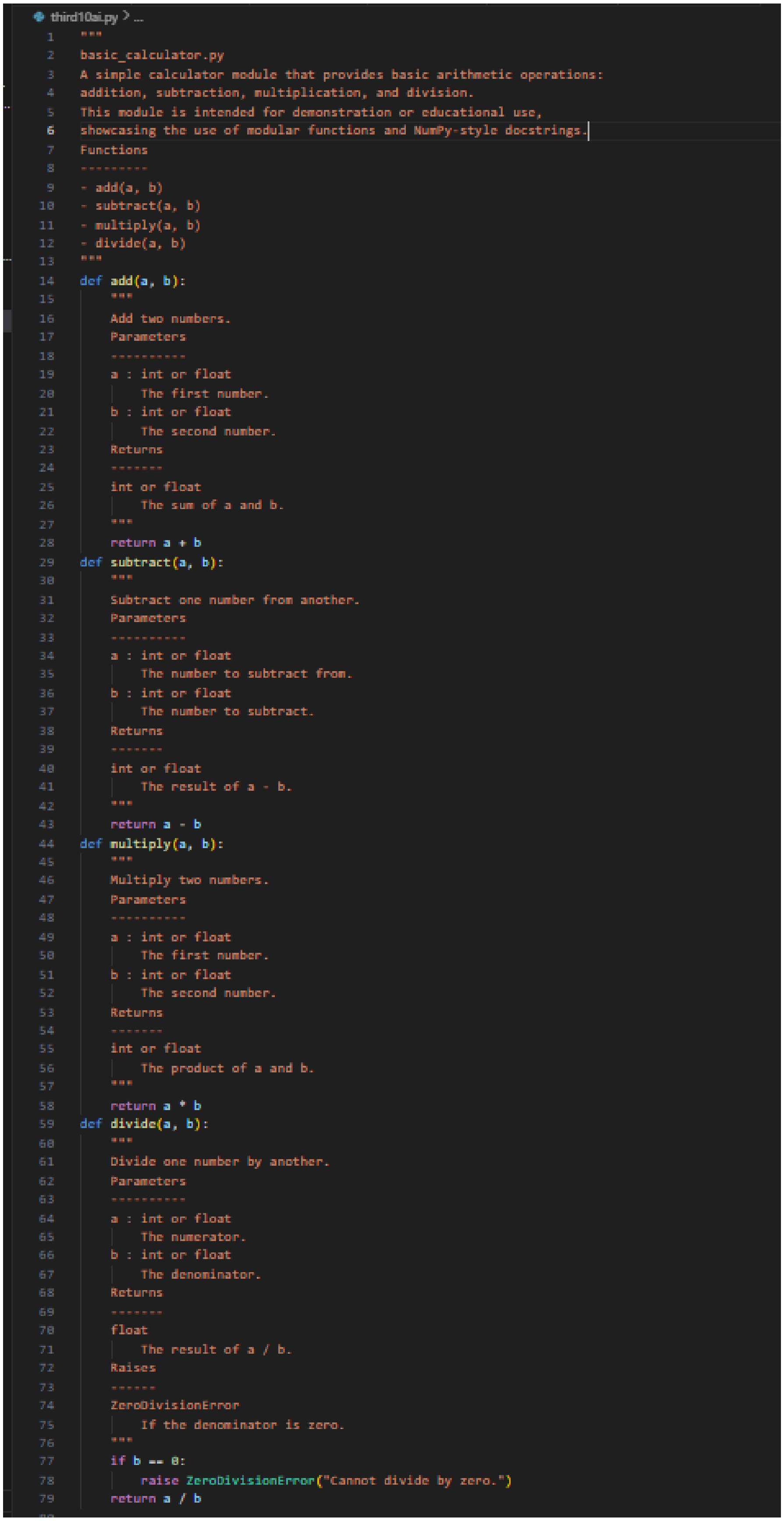
- Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).
- Incorporate manual **docstring** in code with NumPy Style

- Use AI assistance to generate a module-level docstring + individual function docstrings.
- Compare the AI-generated docstring with your manually written one.

MANUAL CODE:



```python
def add(a, b):
    """
    Add two numbers.
    args
    a : The first number.
    b : The second number.
    Returns The sum of a and b.
    """
    return a + b
def subtract(a, b):
    """
    Subtract one number from another.
    args
    a : The number to subtract from.
    b : The number to subtract.
    Returns The result of a - b.
    """
    return a - b
def multiply(a, b):
    """
    Multiply two numbers.
    args
    a : The first number.
    b : The second number.
    Returns The product of a and b.
    """
    return a * b
def divide(a, b):
    """
    Divide one number by another.
    args
    a : The numerator.
    b : The denominator.
    Returns The result of a / b.
    Raises ZeroDivisionError
        If b is zero.
    """
    if b == 0:
        raise ZeroDivisionError("Cannot divide by zero.")
    return a / b
```



```python
"""
basic_calculator.py
A simple calculator module that provides basic arithmetic operations:
addition, subtraction, multiplication, and division.
This module is intended for demonstration or educational use,
showcasing the use of modular functions and NumPy-style docstrings.
Functions
---------
- add(a, b)
- subtract(a, b)
- multiply(a, b)
- divide(a, b)
"""
def add(a, b):
    """
    Add 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
def subtract(a, b):
    """
    Subtract one number from another.
    Parameters
    ----------
    a : int or float
        The number to subtract from.
    b : int or float
        The number to subtract.
    Returns
    -------
    int or float
        The result of a - b.
    """
    return a - b
def multiply(a, b):
    """
    Multiply 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
def divide(a, b):
    """
    Divide one number by another.
    Parameters
    ----------
    a : int or float
        The numerator.
    b : int or float
        The denominator.
    Returns
    -------
    float
        The result of a / b.
    Raises
    ------
    ZeroDivisionError
        If the denominator is zero.
    """
    if b == 0:
        raise ZeroDivisionError("Cannot divide by zero.")
    return a / b
```

OBSERVATION:

- **AI-assisted docstrings** offer:

  - Professional structure
  - Better readability
  - Tool compatibility
  - Reusability in teams, APIs, and documentation websites

- **Manual comments** are a good starting point for beginners, but lack structure and formal style, which may be insufficient for large or shared codebases.

The **manual comments** provide basic understanding but lack structure, standard formatting, and detailed type information. In contrast, the **AI-assisted docstrings** follow the **NumPy documentation standard**, offering clearer structure, better readability, and compatibility with professional tools. This makes them more suitable for collaborative and scalable projects.