AI ASSISTED CODING

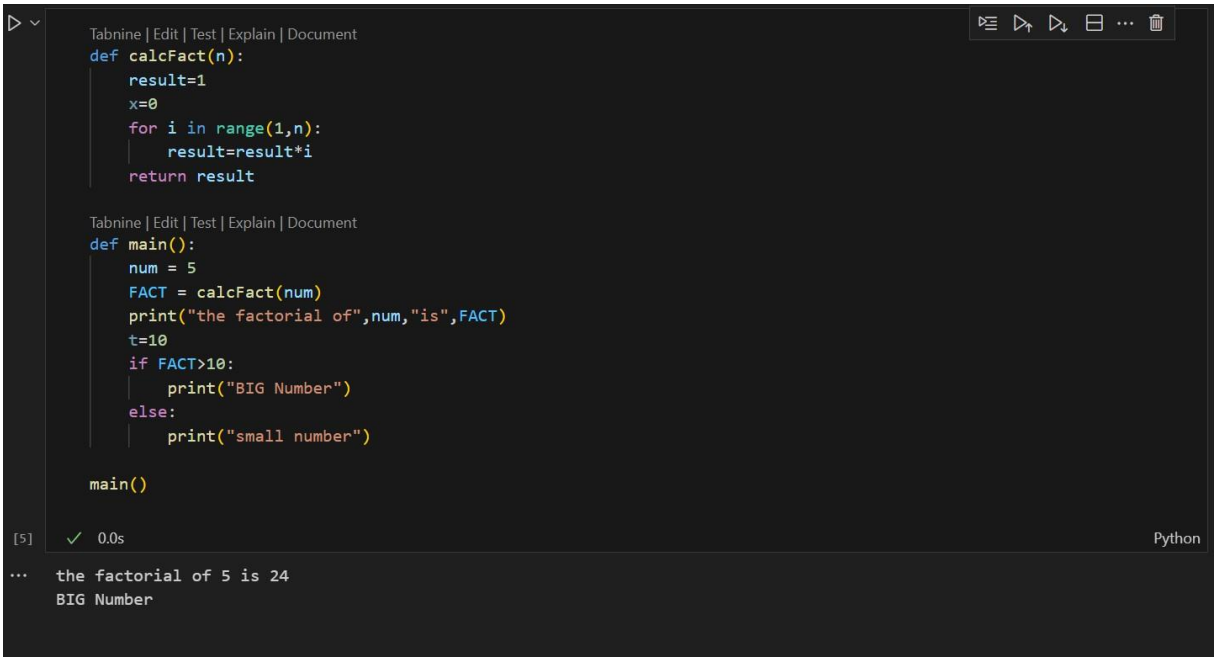
ASSIGNMENT-10.2

A.DHANALAXMI

2403A51269

TASK-1

GIVEN CODE-



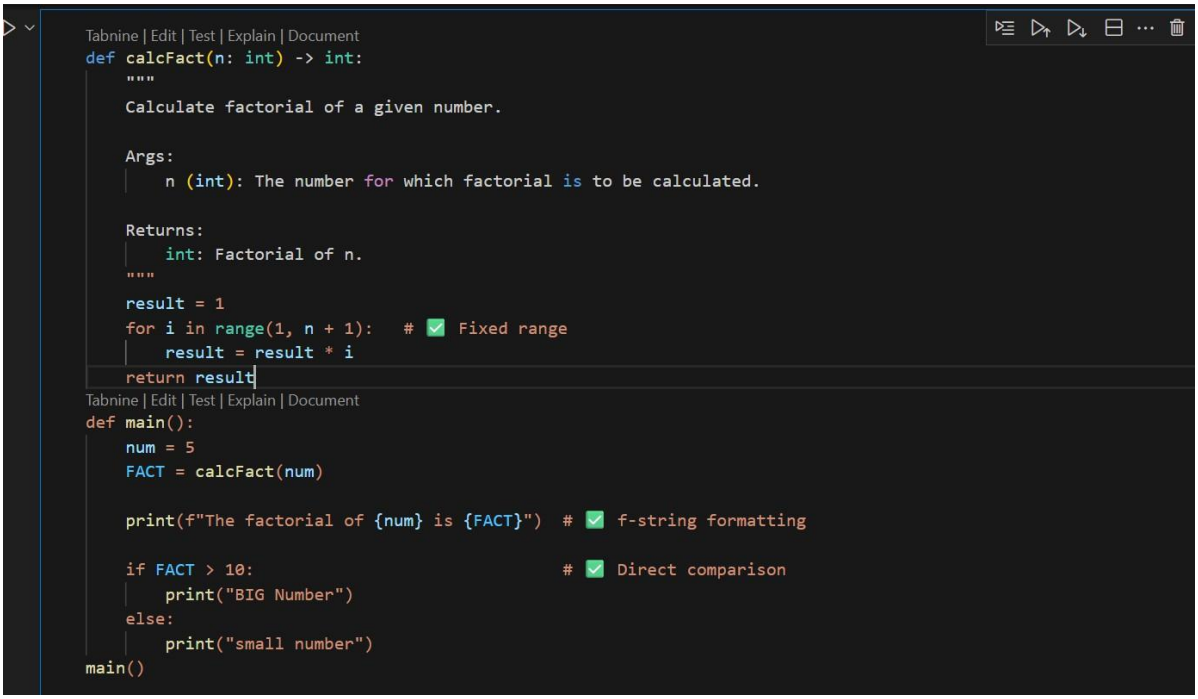
```
Tabnine | Edit | Test | Explain | Document
def calcFact(n):
    result=1
    x=0
    for i in range(1,n):
        result=result*i
    return result

Tabnine | Edit | Test | Explain | Document
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()
```

```
[5]  ✓ 0.0s                                                    Python
...  the factorial of 5 is 24
     BIG Number
```

PROMPT-

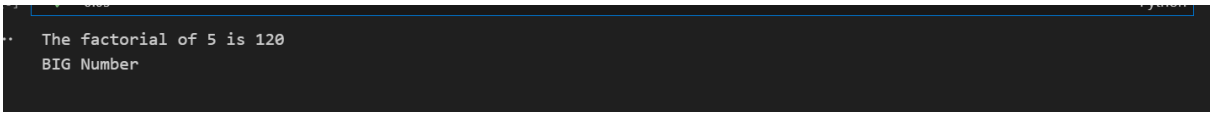
review the above code and suggest correctionS.

AI GENERATED CODE-



```
Tabnine | Edit | Test | Explain | Document
def calcFact(n: int) -> int:
    """
    Calculate factorial of a given number.

    Args:
        n (int): The number for which factorial is to be calculated.

    Returns:
        int: Factorial of n.
    """
    result = 1
    for i in range(1, n + 1):   # ✅ Fixed range
        result = result * i
    return result
Tabnine | Edit | Test | Explain | Document
def main():
    num = 5
    FACT = calcFact(num)

    print(f"The factorial of {num} is {FACT}")  # ✅ f-string formatting

    if FACT > 10:                               # ✅ Direct comparison
        print("BIG Number")
    else:
        print("small number")
main()
```

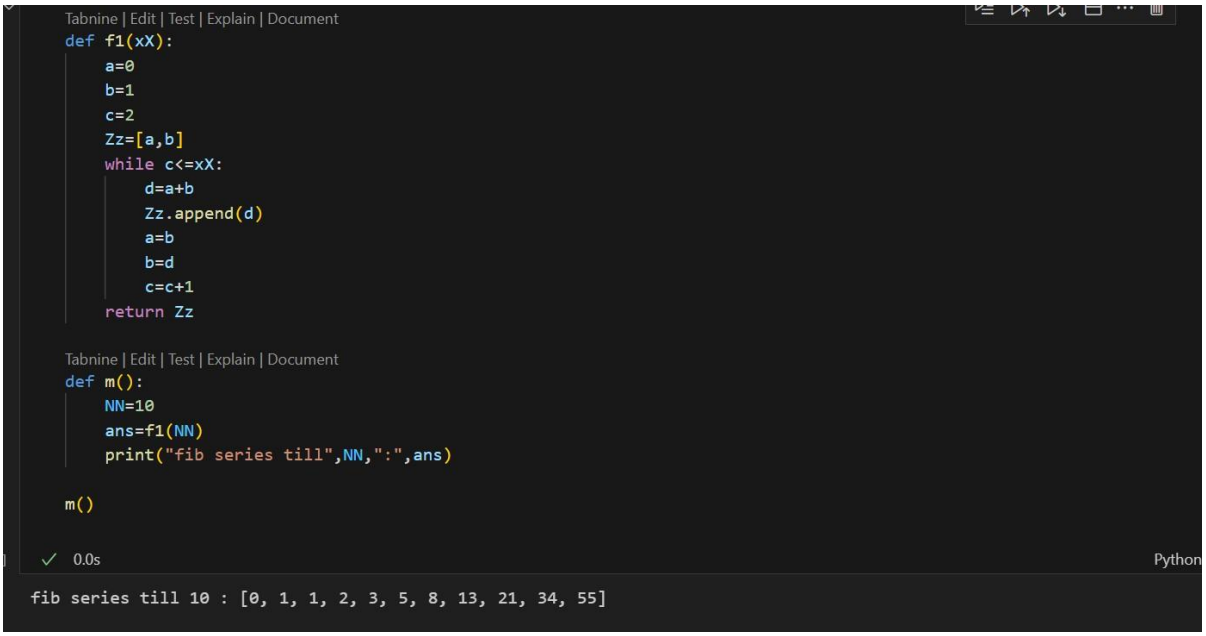
OUTPUT-



```
The factorial of 5 is 120
BIG Number
```

TASK-2

GIVEN CODE-



```
Tabnine | Edit | Test | Explain | Document
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

Tabnine | Edit | Test | Explain | Document
def m():
    NN=10
    ans=f1(NN)
    print("fib series till",NN,":",ans)

m()
```

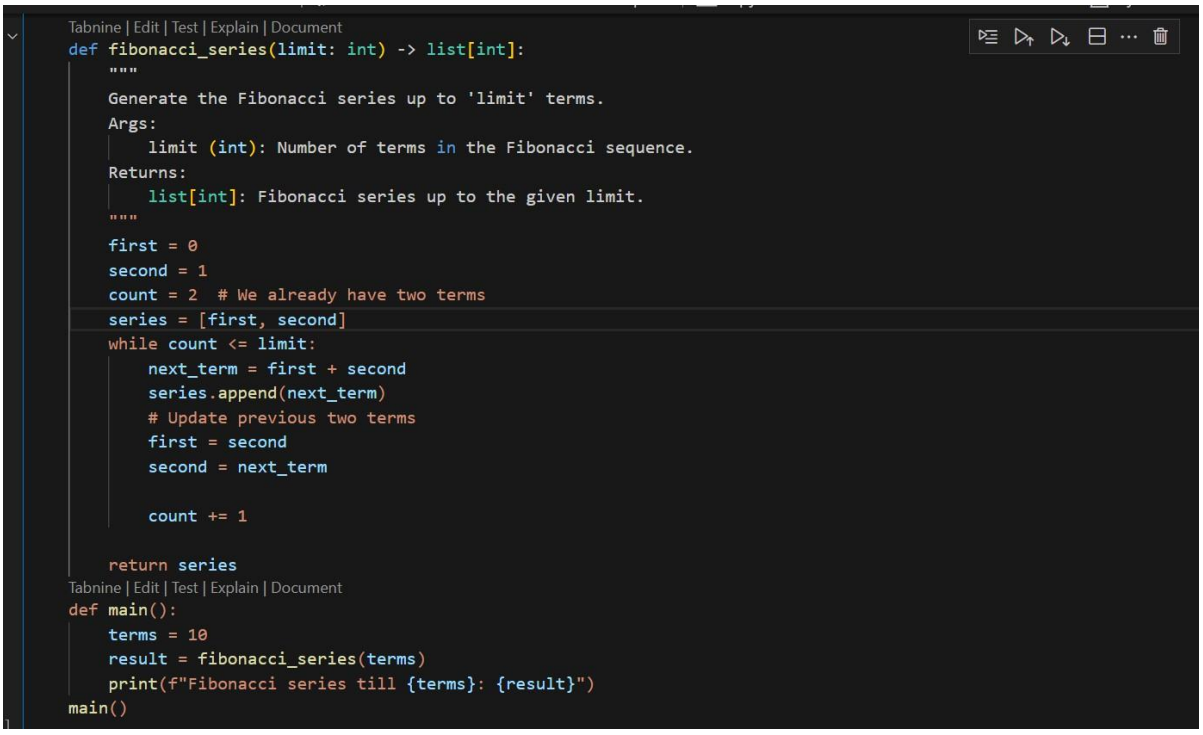
✓ 0.0s Python

```
fib series till 10 : [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

PROMPT-

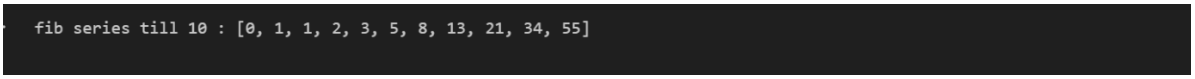
Improve variable names, add comments, and apply PEP8 formatting.

AI GENERATED CODE-



```
Tabnine | Edit | Test | Explain | Document
def fibonacci_series(limit: int) -> list[int]:
    """
    Generate the Fibonacci series up to 'limit' terms.
    Args:
        limit (int): Number of terms in the Fibonacci sequence.
    Returns:
        list[int]: Fibonacci series up to the given limit.
    """
    first = 0
    second = 1
    count = 2  # We already have two terms
    series = [first, second]
    while count <= limit:
        next_term = first + second
        series.append(next_term)
        # Update previous two terms
        first = second
        second = next_term

        count += 1

    return series
Tabnine | Edit | Test | Explain | Document
def main():
    terms = 10
    result = fibonacci_series(terms)
    print(f"Fibonacci series till {terms}: {result}")
main()
```

OUTPUT-



```
fib series till 10 : [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

TASK 3-

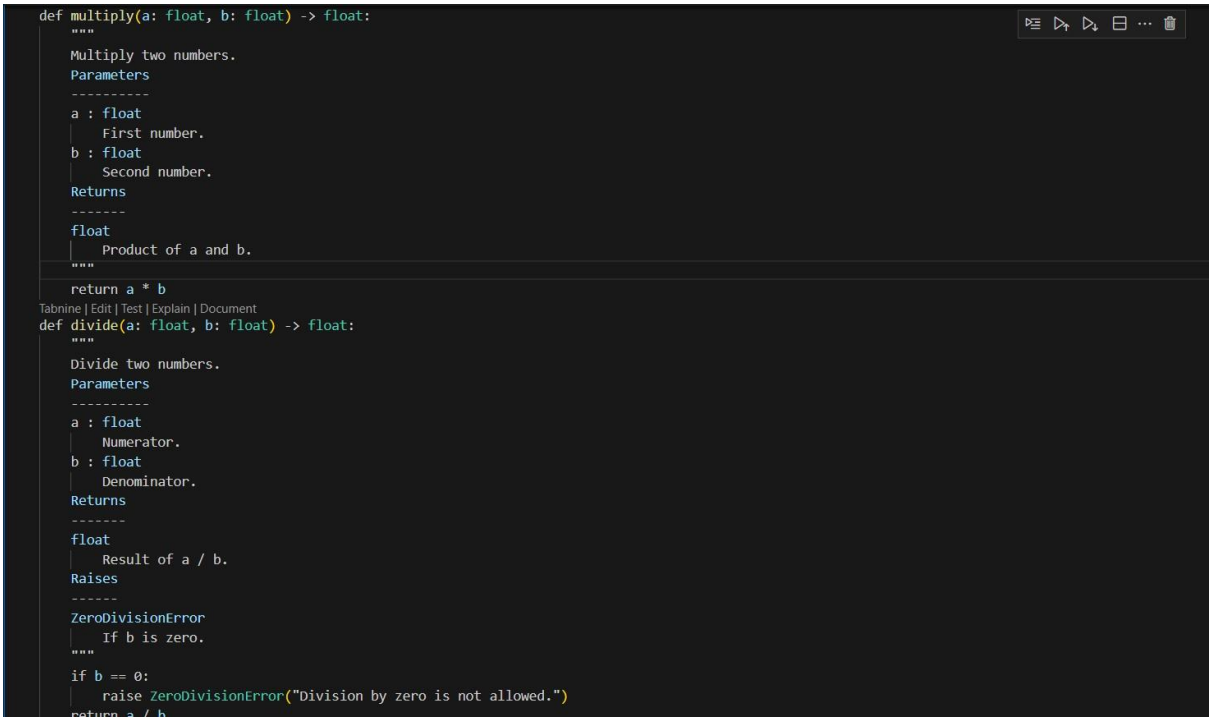
• Write a Python script with 3-4 functions (e.g., calculator: add, subtract, multiply, divide).

• 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.

CODE WRITTEN WITH MANUAL DOCSTRING-

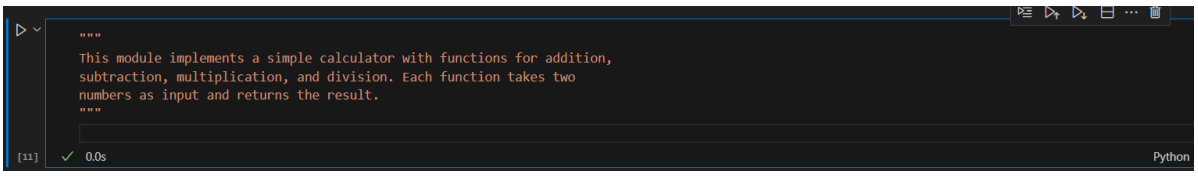


```
def multiply(a: float, b: float) -> float:
    """
    Multiply two numbers.
    Parameters
    ----------
    a : float
        First number.
    b : float
        Second number.
    Returns
    -------
    float
        Product of a and b.
    """
    return a * b
Tabnine | Edit | Test | Explain | Document
def divide(a: float, b: float) -> float:
    """
    Divide two numbers.
    Parameters
    ----------
    a : float
        Numerator.
    b : float
        Denominator.
    Returns
    -------
    float
        Result of a / b.
    Raises
    ------
    ZeroDivisionError
        If b is zero.
    """
    if b == 0:
        raise ZeroDivisionError("Division by zero is not allowed.")
    return a / b
```

PROMPT-

Generate a module-level docstring + individual functiondocstrings for the above code.

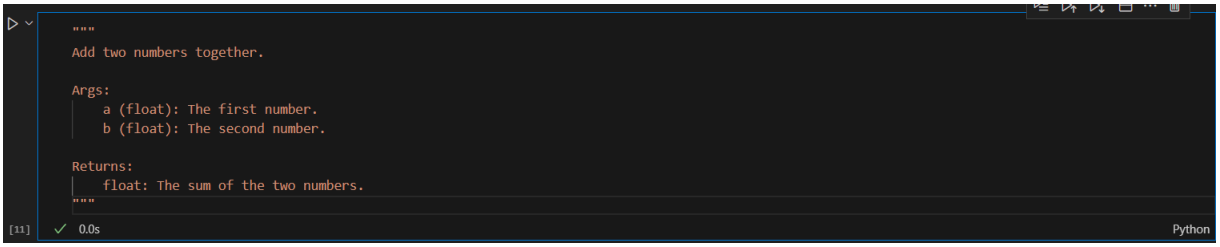
Module-Level Docstring (AI):



```
"""
This module implements a simple calculator with functions for addition,
subtraction, multiplication, and division. Each function takes two
numbers as input and returns the result.
"""
```

[11] ✓ 0.0s Python

Function Docstring (AI Example for add):



COMPARISON-

1. Manual NumPy-style docstrings are detailed and structured with clear sections (Parameters, Returns, Raises).
2. AI-generated docstrings are simpler, shorter, and easier for beginners to read.
3. Manual ones follow professional/academic standards, while AI ones may skip error handling or details.
4. AI saves time, but manual writing ensures accuracy and completeness.