

PROGRAM : B.TECH/CSE

SPECIALIZATION : AIML

COURSE TITLE : AI ASSISTED CODING

COURSE CODE : 24CS101PC214

SEMESTER : 3RD

NAME OF THE STUDENT : KARNAKANTI GODADEVI

ENROLLMENT NO : 2403A52003

BATCH NO : 01

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 this Python code for factorial calculation, identify syntax or logic errors, and suggest corrected and well-commented version using best coding practices.

Code&Output:

The screenshot shows a Google Colab notebook titled 'Untitled8.ipynb'. The code defines a function `calcFact(n)` to calculate the factorial of a non-negative integer `n`. It initializes `result = 1` and iterates from 1 to `n` using a `for` loop, multiplying the result by each integer. The main function `main()` sets `num = 5`, calls `calcFact(num)`, and prints the result. It also sets a threshold `t = 10` and checks if the calculated factorial is greater than the threshold. The code is as follows:

```
# This function calculates the factorial of a non-negative integer n.
def calcFact(n):
    # Initialize the result to 1. The factorial of 0 is 1, and this also serves as the base case for the loop.
    result = 1
    # We iterate from 1 up to and including n.
    # The original code had an off-by-one error in the range, excluding n.
    for i in range(1, n + 1):
        # Multiply the current result by the loop variable i.
        result = result * i
    # Return the calculated factorial.
    return result

# This is the main function where the program execution begins.
def main():
    # Set the number for which to calculate the factorial.
    num = 5
    # Call the calcFact function to get the factorial of num.
    FACT = calcFact(num)
    # Print the result.
    print("The factorial of", num, "is", FACT)

    # Set a threshold for checking the size of the factorial.
    t = 10
    # Check if the calculated factorial is greater than the threshold.
    if FACT > t:
        print("BIG Number")
```

The screenshot shows the same Google Colab notebook after execution. The code is identical to the first screenshot, but the output of the `main()` function is visible at the bottom of the code cell. The output shows that the factorial of 5 is 120, which is greater than the threshold of 10, so it prints "BIG Number".

```
# Set the number for which to calculate the factorial.
num = 5
# Call the calcFact function to get the factorial of num.
FACT = calcFact(num)
# Print the result.
print("The factorial of", num, "is", FACT)

# Set a threshold for checking the size of the factorial.
t = 10
# Check if the calculated factorial is greater than the threshold.
if FACT > t:
    print("BIG Number")
# Check if the calculated factorial is less than the threshold.
elif FACT < t:
    print("small number")
# If the factorial is not greater than or less than the threshold, it must be equal.
else:
    print("Number is equal to threshold")

# Call the main function to start the program.
main()

The factorial of 5 is 120
BIG Number
```

Observation:

- The code defines a function `calcFact` to compute the factorial of a non-negative integer `n`.
- It uses a loop to multiply numbers from 1 up to `n` to get the factorial.
- The main function sets a number (5) and calculates its factorial using `calcFact`.

- It then compares the calculated factorial to a threshold (10) and prints "BIG Number", "small number", or "Number is equal to threshold".
- The output shows the factorial of 5 is 120 and correctly identifies it as a "BIG Number" because 120 is greater than 10.

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.

```
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)

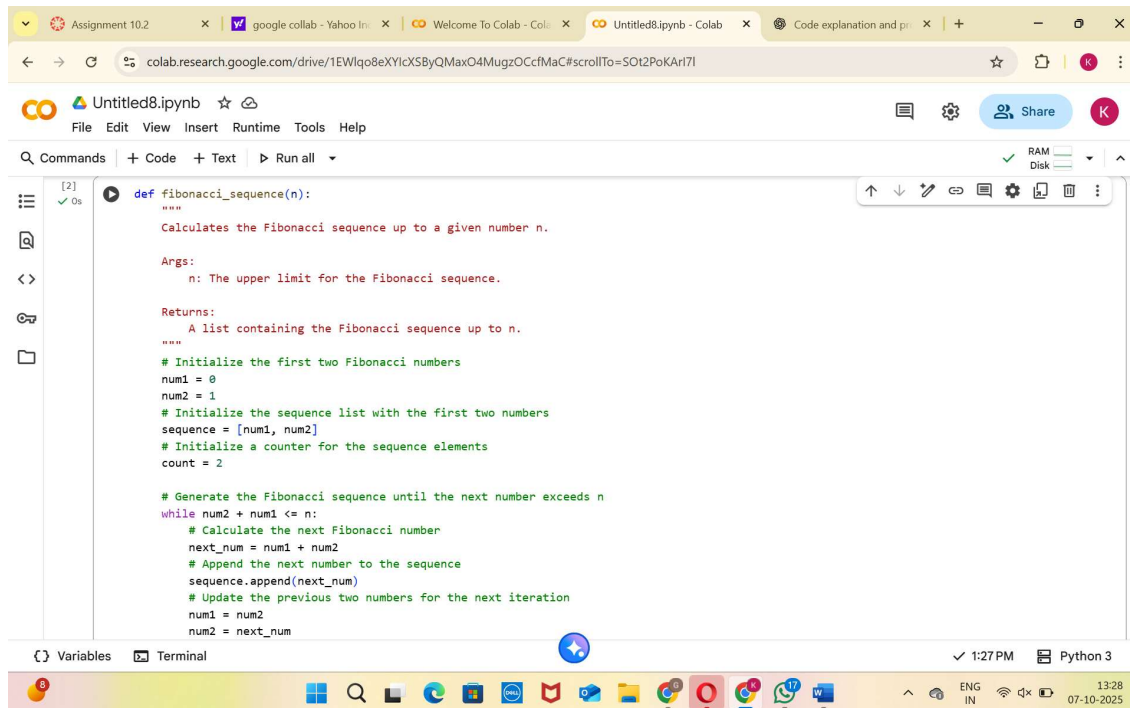
m()
```

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

Prompt:

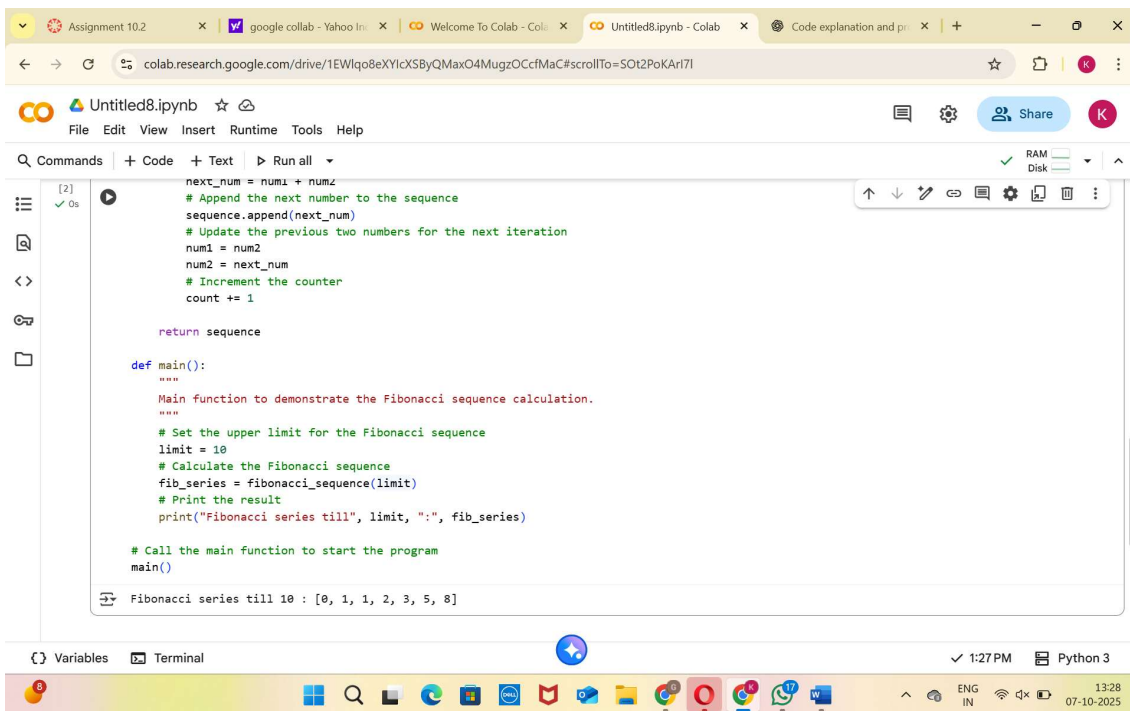
Review this Fibonacci Python code for readability and logic. Apply PEP8 formatting, rename unclear variables, add inline comments, and include docstrings for clarity.

Code&Output:



The screenshot shows a Google Colab notebook titled 'Untitled8.ipynb'. The code defines a function `fibonacci_sequence(n)` that calculates the Fibonacci sequence up to a given number `n`. The function includes docstrings for its purpose, arguments, and return values. It initializes `num1 = 0`, `num2 = 1`, and a `sequence` list with `[num1, num2]`. A `while` loop generates the sequence until the next number exceeds `n`, updating `num1` and `num2` and appending the next number to the `sequence` list. The function returns the `sequence` list.

```
def fibonacci_sequence(n):  
    """  
    Calculates the Fibonacci sequence up to a given number n.  
  
    Args:  
        n: The upper limit for the Fibonacci sequence.  
  
    Returns:  
        A list containing the Fibonacci sequence up to n.  
    """  
    # Initialize the first two Fibonacci numbers  
    num1 = 0  
    num2 = 1  
    # Initialize the sequence list with the first two numbers  
    sequence = [num1, num2]  
    # Initialize a counter for the sequence elements  
    count = 2  
  
    # Generate the Fibonacci sequence until the next number exceeds n  
    while num2 + num1 <= n:  
        # Calculate the next Fibonacci number  
        next_num = num1 + num2  
        # Append the next number to the sequence  
        sequence.append(next_num)  
        # Update the previous two numbers for the next iteration  
        num1 = num2  
        num2 = next_num  
  
    return sequence
```



The screenshot shows the same Google Colab notebook with the `main` function defined. The `main` function sets a `limit = 10`, calls `fibonacci_sequence(limit)` to calculate the sequence, and prints the result. The output of the program is displayed at the bottom: `Fibonacci series till 10 : [0, 1, 1, 2, 3, 5, 8]`.

```
def main():  
    """  
    Main function to demonstrate the Fibonacci sequence calculation.  
    """  
    # Set the upper limit for the Fibonacci sequence  
    limit = 10  
    # Calculate the Fibonacci sequence  
    fib_series = fibonacci_sequence(limit)  
    # Print the result  
    print("Fibonacci series till", limit, ":", fib_series)  
  
    # Call the main function to start the program  
    main()
```

Fibonacci series till 10 : [0, 1, 1, 2, 3, 5, 8]

Observation:

- The code defines a function `fibonacci_sequence` that calculates the Fibonacci sequence up to a given limit `n`.
- It initializes the first two Fibonacci numbers (0 and 1) and a list to store the sequence.

- A while loop iteratively calculates and appends the next Fibonacci number to the sequence until the next number exceeds the limit.
- The main function demonstrates how to call fibonacci_sequence and prints the resulting series.
- The output shows the Fibonacci numbers generated up to the specified limit (10 in this case), which are [0, 1, 1, 2, 3, 5, 8].

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:

Generate a module-level and function-level docstring for my Python calculator script using NumPy style. Ensure clear descriptions, parameters, return values, and exception handling. Also, check for code smells and suggest refactoring improvements.

Code&Output:

Assignment 10.2 | google colab - Yahoo Inc | Welcome To Colab - Colab | Untitled8.ipynb - Colab | Code explanation and p | +

colab.research.google.com/drive/1EWlqo8eXYlcXSByQMaxO4MugzOCcfMaC#scrollTo=hwK-mvbtSNLs

Untitled8.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Fibonacci series till 10 : [0, 1, 1, 2, 3, 5, 8]

```
[3] ✓ 0s
"""
A simple calculator module for basic arithmetic operations.

This module provides functions for addition, subtraction, multiplication,
and division of two numbers.
"""

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 the two numbers.
    """
    return a + b
```

Variables Terminal

1:32 PM Python 3

Assignment 10.2 | google colab - Yahoo Inc | Welcome To Colab - Colab | Untitled8.ipynb - Colab | Code explanation and p | +

colab.research.google.com/drive/1EWlqo8eXYlcXSByQMaxO4MugzOCcfMaC#scrollTo=hwK-mvbtSNLs

Untitled8.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[3] ✓ 0s
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 the two numbers.
    """
    return a - b

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

    Parameters
    -----
    a : int or float
        The first number.
```

Variables Terminal

1:32 PM Python 3

Assignment 10.2 | google colab - Yahoo Inc | Welcome To Colab - Colab | Untitled8.ipynb - Colab | Code explanation and p | + | - | X

colab.research.google.com/drive/1EWIqo8eXYIcXSByQMaxO4MugzOCcfMaC#scrollTo=hwK-mvbtSNLs

Untitled8.ipynb | File Edit View Insert Runtime Tools Help

Commands | + Code | + Text | ▶ Run all

RAM Disk

```
[3] ✓ 0s
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 the two numbers.
    """
    return a - b

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

    Parameters
    -----
    a : int or float
        The first number.
```

Variables | Terminal

1:32 PM Python 3

Assignment 10.2 | google colab - Yahoo Inc | Welcome To Colab - Colab | Untitled8.ipynb - Colab | Code explanation and p | + | - | X

colab.research.google.com/drive/1EWIqo8eXYIcXSByQMaxO4MugzOCcfMaC#scrollTo=hwK-mvbtSNLs

Untitled8.ipynb | File Edit View Insert Runtime Tools Help

Commands | + Code | + Text | ▶ Run all

RAM Disk

```
[3] ✓ 0s
    Parameters
    -----
    a : int or float
        The first number.
    b : int or float
        The second number.

    Returns
    -----
    int or float
        The product of the two numbers.
    """
    return a * b

def divide(a, b):
    """
    Divides the first number by the second.

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

    Returns
    -----
```

Variables | Terminal

1:32 PM Python 3

```
[3] ✓ Os
print(f"{num1} + {num2} = {add(num1, num2)}")
print(f"{num1} - {num2} = {subtract(num1, num2)}")
print(f"{num1} * {num2} = {multiply(num1, num2)}")

try:
    print(f"{num1} / {num2} = {divide(num1, num2)}")
    print(f"{num1} / 0 = {divide(num1, 0)}")
except ZeroDivisionError as e:
    print(f"Error: {e}")
```

10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
Error: Division by zero is not allowed

Observation:

- The code defines functions for basic arithmetic operations: addition, subtraction, multiplication, and division.
- It includes clear NumPy-style docstrings explaining each function's purpose, parameters, return values, and potential exceptions.
- The divide function explicitly checks for division by zero and raises a `ZeroDivisionError`.
- The example usage demonstrates how to use the functions and includes a `try...except` block to handle the `ZeroDivisionError` gracefully.
- The `if __name__ == "__main__":` block ensures the example usage only runs when the script is executed directly.