AI ASSISTED CODING

NAME: NAREDDY JASMITHA

ENROLLMENT NO : 2403A51281

COURSE : AI ASSISTED CODING

SEM :3RD

BRANCH : CSE/AIML

DATE: 07-10-2025

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()
```

PROMPT: Develop a code to generate 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

```python
def calcFact(n):
    # Initialize result to 1, as the factorial of 0 is 1, and it's the multiplicative identity.
    result = 1
    # The variable 'x' is not used, so it's best to remove it.

    # The range function should be range(1, n + 1) to include 'n' itself in the multiplication.
    # range(1, n) only goes up to n-1.
    for i in range(1, n + 1):
        result = result * i

    return result

def main():
    # Get user input for the number
    num_str = input("Enter a non-negative integer: ")
    try:
        num = int(num_str)
        if num < 0:
            print("Factorial is not defined for negative numbers.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    FACT = calcFact(num)

    # Use an f-string (formatted string literal) for clearer and more conventional printing.
    print(f"the factorial of {num} is {FACT}")
```

```
        # Use an f-string (formatted string literal) for clearer and more conventional printing.
        print(f"the factorial of {num} is {FACT}")

        # The variable 't' is declared but not used in the following 'if' statement,
        # so it's best to remove it unless it was intended for something else.

        # The conditional logic is correct: 120 is greater than 10.
        if FACT > 10:
            print("BIG Number")
        else:
            print("small number")

# It's a standard practice to call the main function inside an 'if __name__ == "__main__":' block.
# This ensures that 'main()' only runs when the script is executed directly.
if __name__ == "__main__":
    main()
```

```
Enter a non-negative integer: 23
the factorial of 23 is 25852016738884976640000
BIG Number
```

EXPLANATION:

1. `calcFact(n)` function:
    - Takes an integer `n` as input.
    - Initializes a `result` variable to 1.
    - Uses a `for` loop to iterate from 1 up to and including `n`.
    - In each iteration, it multiplies the current `result` by the loop variable `i`.
    - Returns the final calculated factorial `result`.
2. **`main()` function:**
    - Prompts the user to "Enter a non-negative integer:" using the `input()` function.
    - Uses a `try-except` block to handle potential `ValueError` if the user enters something that is not an integer.
    - If the input is valid and non-negative, it calls the `calcFact()` function with the user's number to get the factorial.
    - Prints the calculated factorial using an f-string for clear formatting.
    - Checks if the calculated factorial (`FACT`) is greater than 10 and prints "BIG Number" or "small number" accordingly.

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()
```

PROMPT: Develop a python code to generate a Python code for Fibonacci as
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.

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

```
def f1(x_limit):
    a = 0
    b = 1
    c = 2
    zz = [a, b]

    while c <= x_limit:
        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()
```

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

EXPLANATION:

Certainly! Here is an explanation of the selected code cell:

This code calculates and prints a Fibonacci series up to a given limit.

- The `f1(x_limit)` function generates the Fibonacci series. It initializes the first two numbers of the series (`a` and `b`) and a list `zz` with these numbers. It then iteratively calculates the next number in the series by adding the previous two, appends it to the list, and updates the previous two numbers until the limit `x_limit` is reached.
- The `m()` function sets the limit for the Fibonacci series to 10, calls `f1()` to get the series, and then prints the series.
- The final line `m()` calls the `m()` function to execute the code.

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.

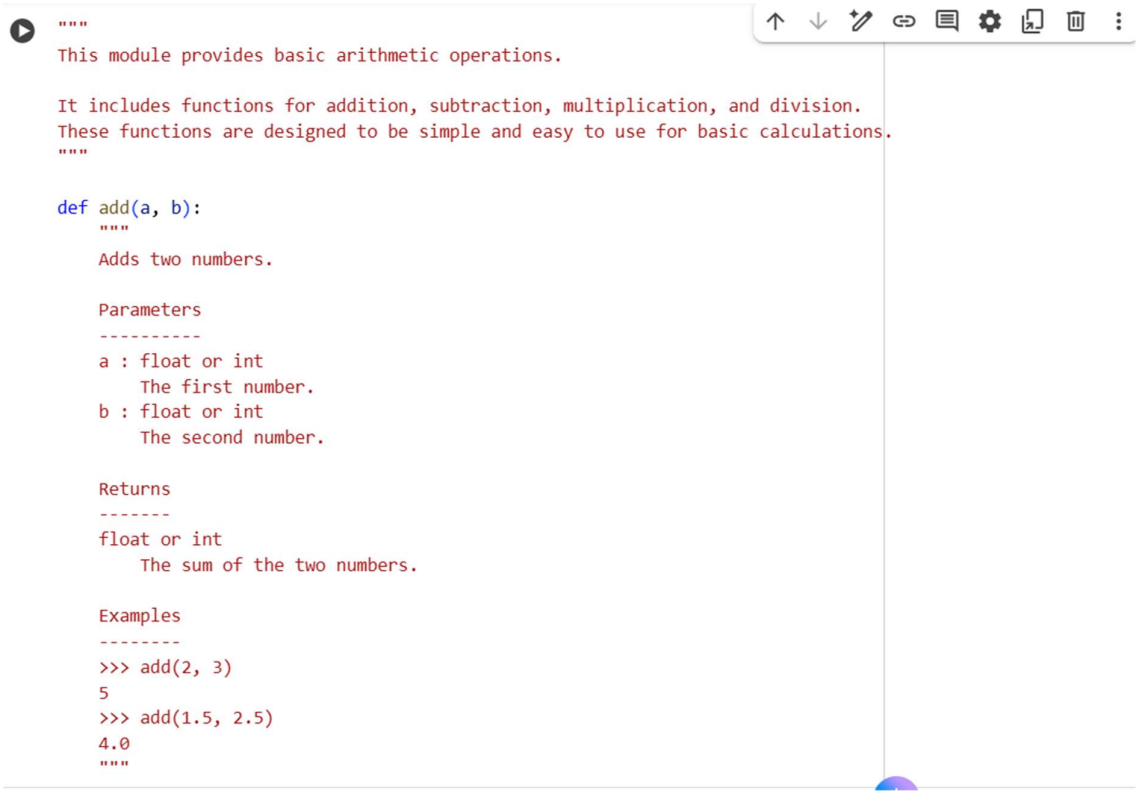
• Poor Naming – Variables or functions with confusing names (x1, foo, data123).
• Unused Variables – Declaring variables but never using them.
• Magic Numbers – Using unexplained constants (3.14159 instead of PI).
• Deep Nesting – Too many if/else levels, making code hard to read.
• Large Class – A single 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

PROMPT:

Generate a python code to develop 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



```
"""
This module provides basic arithmetic operations.

It includes functions for addition, subtraction, multiplication, and division.
These functions are designed to be simple and easy to use for basic calculations.
"""

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

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

    Returns
    -------
    float or int
        The sum of the two numbers.

    Examples
    --------
    >>> add(2, 3)
    5
    >>> add(1.5, 2.5)
    4.0
    """
```

```
        4.0
        """
        return a + b

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

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

        Returns
        -------
        float or int
            The difference between the two numbers.

        Examples
        --------
        >>> subtract(5, 2)
        3
        >>> subtract(10.0, 3.5)
        6.5
        """
        return a - b

    def multiply(a, b):
```

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

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

    Returns
    -------
    float or int
        The product of the two numbers.

    Examples
    --------
    >>> multiply(4, 5)
    20
    >>> multiply(2.5, 2.0)
    5.0
    """
    return a * b

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

    Parameters
    ----------
```

```
    Divides the first number by the second.

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

    Returns
    -------
    float or int
        The quotient of the division.

    Raises
    ------
    ValueError
        If the denominator is zero.

    Examples
    --------
    >>> divide(10, 2)
    5.0
    >>> divide(5.0, 2.5)
    2.0
    """
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
```

```
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

# Example usage:
# print(add(5, 3))
# print(subtract(10, 4))
# print(multiply(6, 7))
# print(divide(20, 5))
# print(divide(10, 0)) # This will raise a ValueError
```

EXPLANATION:

- **Module Docstring**: The triple-quoted string at the beginning is a module-level docstring. It explains the overall purpose of the module.
- **Functions**:
    - `add(a, b)`: Takes two numbers (`a` and `b`) and returns their sum.
    - `subtract(a, b)`: Takes two numbers (`a` and `b`) and returns the result of subtracting `b` from `a`.
    - `multiply(a, b)`: Takes two numbers (`a` and `b`) and returns their product.
    - `divide(a, b)`: Takes two numbers (`a` and `b`) and returns the result of dividing `a` by `b`. It includes error handling to raise a `ValueError` if the denominator (`b`) is zero.
- **Function Docstrings**: Each function has a docstring following the NumPy style. These docstrings explain:
    - The purpose of the function.
    - The parameters it accepts (`Parameters` section) with their

- **Function Docstrings**: Each function has a docstring following the NumPy style. These docstrings explain:
    - The purpose of the function.
    - The parameters it accepts (`Parameters` section) with their types and descriptions.
    - The value it returns (`Returns` section) with its type and description.
    - Examples of how to use the function (`Examples` section).
- **Example Usage**: The commented-out lines at the end show how to call each function.

his code is well-documented with clear explanations of each function and ıe module as a whole, following the specified docstring style.