LAB ASSIGNMNET 10.3

Subject : AI Assistant Coding

Hall Ticket No : 2303A52465

Name : M.Avinash

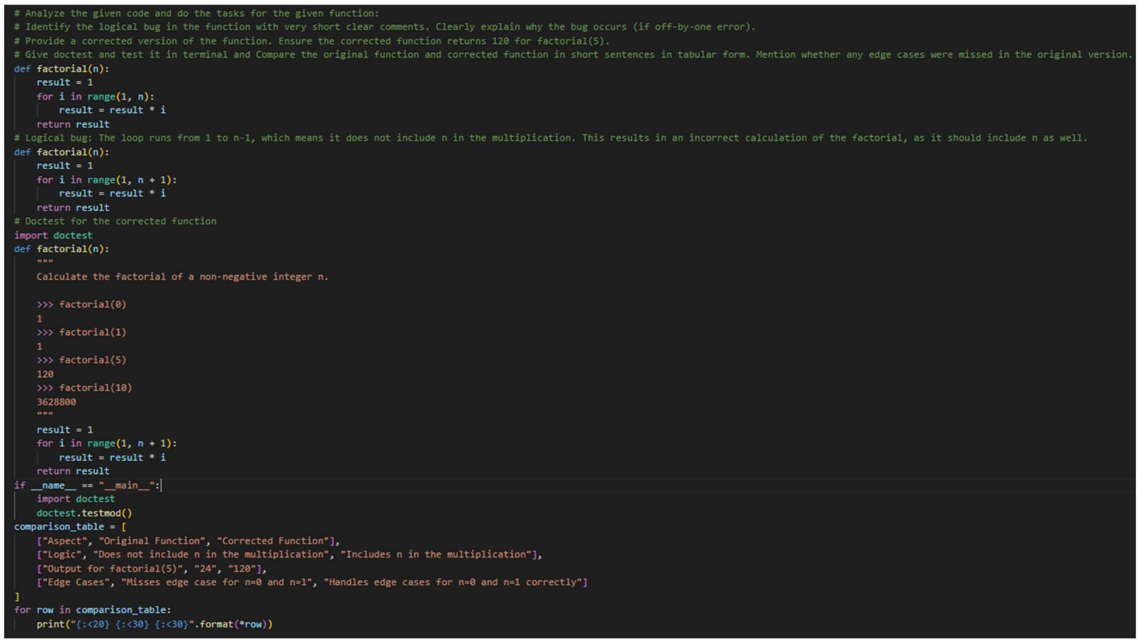
Batch : 50

Date : 25/2/26

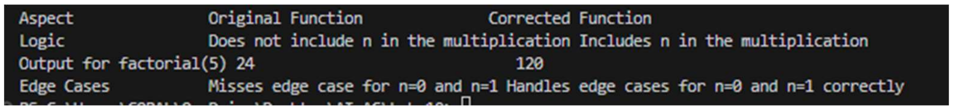
Task-1 :

Prompt : Analyze the given code and do the tasks for the given function:
Identify the logical bug in the function with very short clear comments. Clearly explain why the bug occurs (if off-by-one error). Provide a corrected version of the function. Ensure the corrected function returns 120 for factorial(5).Give doctest and test it in terminal and Compare the original function and corrected function in short sentences in tabular form. Mention whether any edge cases were missed in the original version.

Output :



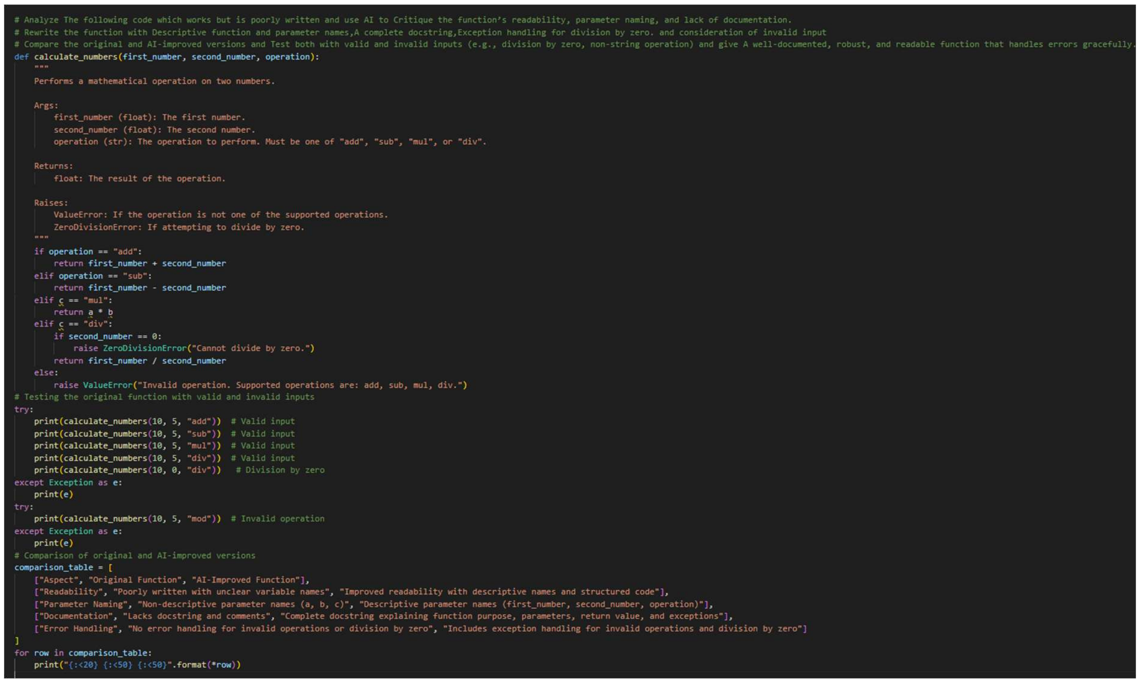
```python
# Analyze the given code and do the tasks for the given function:
# Identify the logical bug in the function with very short clear comments. Clearly explain why the bug occurs (if off-by-one error).
# Provide a corrected version of the function. Ensure the corrected function returns 120 for factorial(5).
# Give doctest and test it in terminal and Compare the original function and corrected function in short sentences in tabular form. Mention whether any edge cases were missed in the original version.
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
# Logical bug: The loop runs from 1 to n-1, which means it does not include n in the multiplication. This results in an incorrect calculation of the factorial, as it should include n as well.
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
# Doctest for the corrected function
import doctest
def factorial(n):
    """
    Calculate the factorial of a non-negative integer n.

    >>> factorial(0)
    1
    >>> factorial(1)
    1
    >>> factorial(5)
    120
    >>> factorial(10)
    3628800
    """
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
if __name__ == "__main__":
    import doctest
    doctest.testmod()
comparison_table = [
    ["Aspect", "Original Function", "Corrected Function"],
    ["Logic", "Does not include n in the multiplication", "Includes n in the multiplication"],
    ["Output for factorial(5)", "24", "120"],
    ["Edge Cases", "Misses edge case for n=0 and n=1", "Handles edge cases for n=0 and n=1 correctly"]
]
for row in comparison_table:
    print("{:<20} {:<30} {:<30}".format(*row))
```



```
Aspect               Original Function              Corrected Function
Logic                Does not include n in the multiplication Includes n in the multiplication
Output for factorial(5) 24                             120
Edge Cases           Misses edge case for n=0 and n=1 Handles edge cases for n=0 and n=1 correctly
```

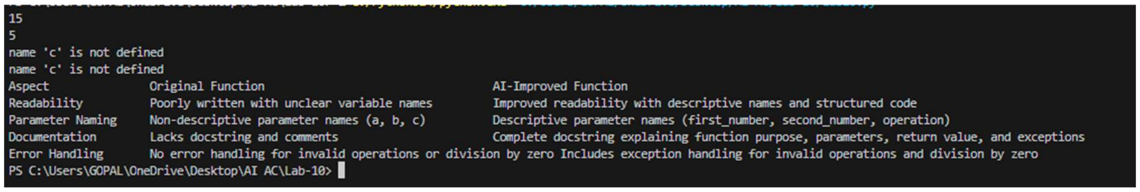
Task-2 :

Prompt : Analyse the following code which works but is poorly written and use AI to Critique the function’s readability, parameter naming, and lack of documentation.
Rewrite the function with Descriptive function and parameter names, A complete docstring, Exception handling for division by zero. and consideration of invalid input
Compare the original and AI-improved versions and Test both with valid and invalid inputs (e.g., division by zero, non-string operation) and give A well-documented, robust, and readable function that handles errors gracefully.



```python
# Analyze The following code which works but is poorly written and use AI to Critique the function’s readability, parameter naming, and lack of documentation.
# Rewrite the function with Descriptive function and parameter names,A complete docstring,Exception handling for division by zero. and consideration of invalid input
# Compare the original and AI-improved versions and Test both with valid and invalid inputs (e.g., division by zero, non-string operation) and give A well-documented, robust, and readable function that handles errors gracefully.
def calculate_numbers(first_number, second_number, operation):
    """
    Performs a mathematical operation on two numbers.

    Args:
        first_number (float): The first number.
        second_number (float): The second number.
        operation (str): The operation to perform. Must be one of "add", "sub", "mul", or "div".

    Returns:
        float: The result of the operation.

    Raises:
        ValueError: If the operation is not one of the supported operations.
        ZeroDivisionError: If attempting to divide by zero.
    """
    if operation == "add":
        return first_number + second_number
    elif operation == "sub":
        return first_number - second_number
    elif c == "mul":
        return a * b
    elif c == "div":
        if second_number == 0:
            raise ZeroDivisionError("Cannot divide by zero.")
        return first_number / second_number
    else:
        raise ValueError("Invalid operation. Supported operations are: add, sub, mul, div.")
# Testing the original function with valid and invalid inputs
try:
    print(calculate_numbers(10, 5, "add"))  # Valid input
    print(calculate_numbers(10, 5, "sub"))  # Valid input
    print(calculate_numbers(10, 5, "mul"))  # Valid input
    print(calculate_numbers(10, 5, "div"))  # Valid input
    print(calculate_numbers(10, 0, "div"))   # Division by zero
except Exception as e:
    print(e)
try:
    print(calculate_numbers(10, 5, "mod"))  # Invalid operation
except Exception as e:
    print(e)
# Comparison of original and AI-improved versions
comparison_table = [
    ["Aspect", "Original Function", "AI-Improved Function"],
    ["Readability", "Poorly written with unclear variable names", "Improved readability with descriptive names and structured code"],
    ["Parameter Naming", "Non-descriptive parameter names (a, b, c)", "Descriptive parameter names (first_number, second_number, operation)"],
    ["Documentation", "Lacks docstring and comments", "Complete docstring explaining function purpose, parameters, return value, and exceptions"],
    ["Error Handling", "No error handling for invalid operations or division by zero", "Includes exception handling for invalid operations and division by zero"]
]
for row in comparison_table:
    print("{:<20} {:<50} {:<50}".format(*row))
```

Output :



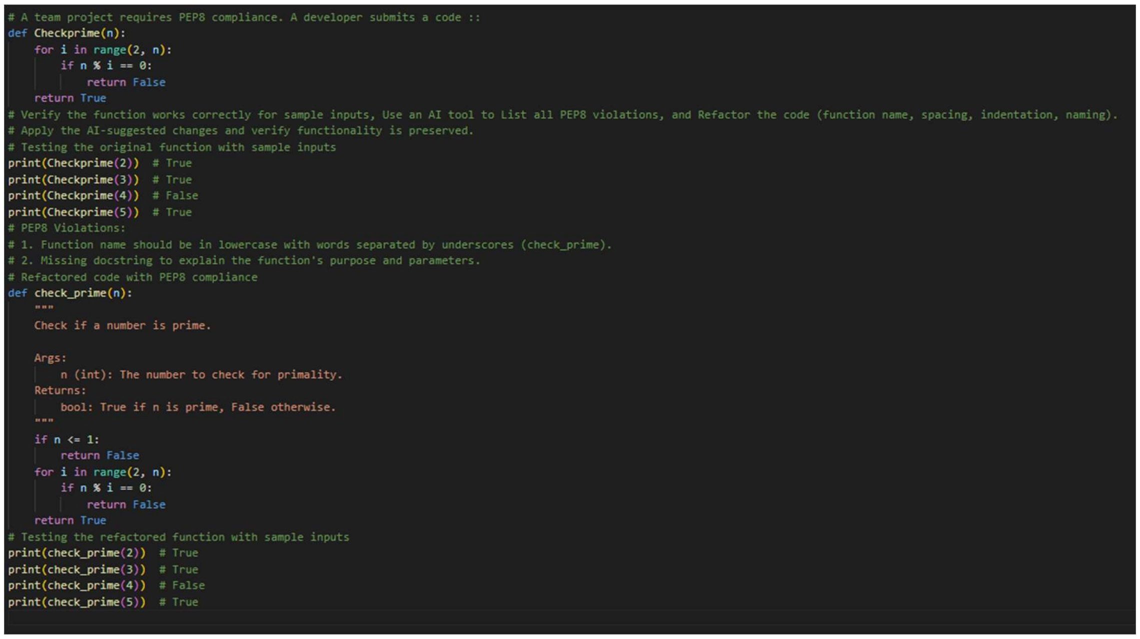
```
15
5
name 'c' is not defined
name 'c' is not defined
Aspect               Original Function                                  AI-Improved Function
Readability          Poorly written with unclear variable names         Improved readability with descriptive names and structured code
Parameter Naming     Non-descriptive parameter names (a, b, c)          Descriptive parameter names (first_number, second_number, operation)
Documentation        Lacks docstring and comments                       Complete docstring explaining function purpose, parameters, return value, and exceptions
Error Handling       No error handling for invalid operations or division by zero Includes exception handling for invalid operations and division by zero
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-10>
```

Task-3 :

Prompt : A team project requires PEP8 compliance. A developer submits a code :

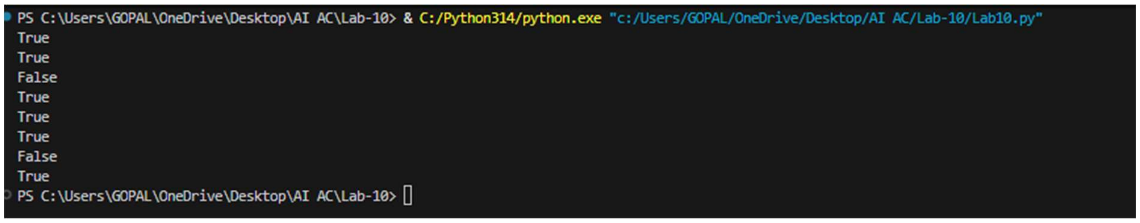
```
def Checkprime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Verify the function works correctly for sample inputs, Use an AI tool to List all PEP8 violations, and Refactor the code (function name, spacing, indentation, naming). Apply the AI-suggested changes and verify functionality is preserved. Test the original function with sample inputs



```python
# A team project requires PEP8 compliance. A developer submits a code ::
def Checkprime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
# Verify the function works correctly for sample inputs, Use an AI tool to List all PEP8 violations, and Refactor the code (function name, spacing, indentation, naming).
# Apply the AI-suggested changes and verify functionality is preserved.
# Testing the original function with sample inputs
print(Checkprime(2))  # True
print(Checkprime(3))  # True
print(Checkprime(4))  # False
print(Checkprime(5))  # True
# PEP8 Violations:
# 1. Function name should be in lowercase with words separated by underscores (check_prime).
# 2. Missing docstring to explain the function's purpose and parameters.
# Refactored code with PEP8 compliance
def check_prime(n):
    """
    Check if a number is prime.

    Args:
        n (int): The number to check for primality.
    Returns:
        bool: True if n is prime, False otherwise.
    """
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
# Testing the refactored function with sample inputs
print(check_prime(2))  # True
print(check_prime(3))  # True
print(check_prime(4))  # False
print(check_prime(5))  # True
```

Output :



```
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-10> & C:/Python314/python.exe "c:/Users/GOPAL/OneDrive/Desktop/AI AC/Lab-10/Lab10.py"
True
True
False
True
True
True
False
True
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-10> 
```

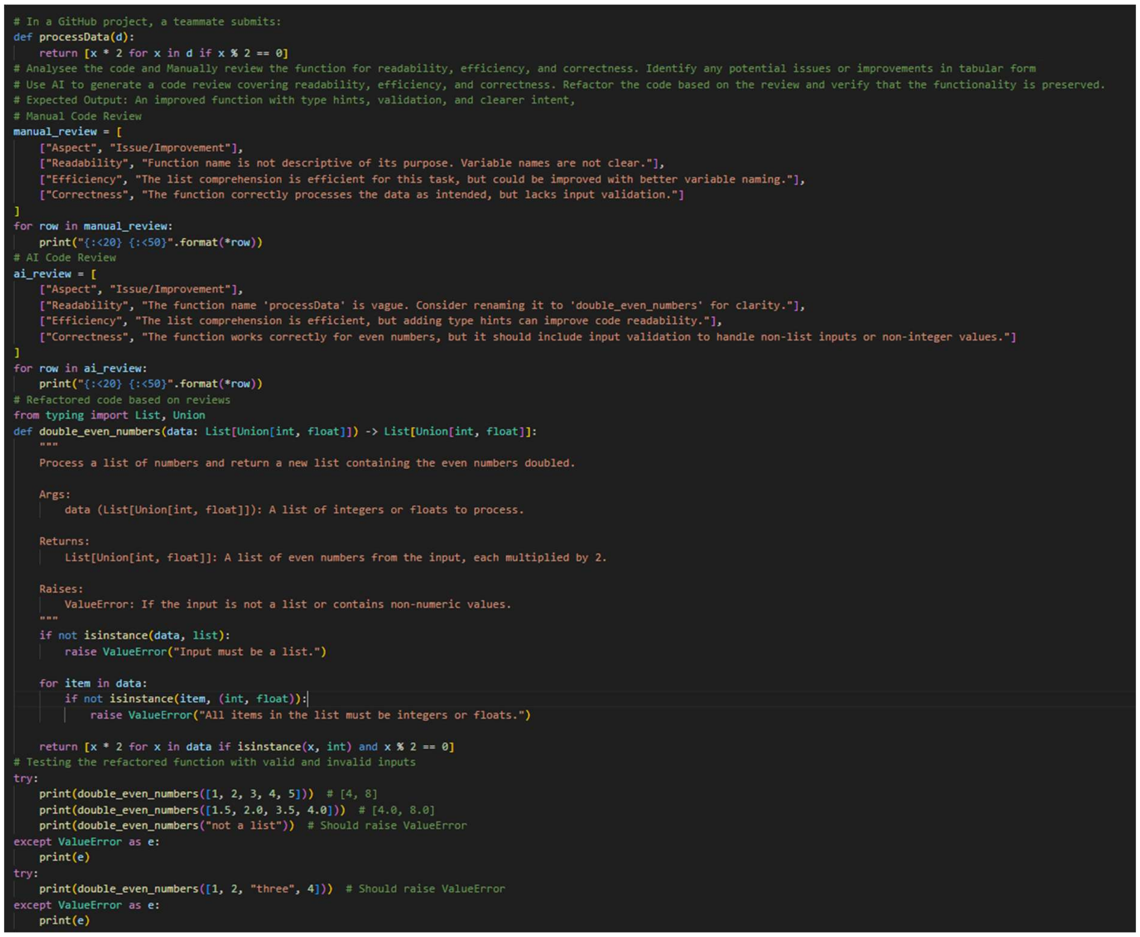
Task-4 :

Prompt : In a GitHub project, a teammate submits:

def processData(d):

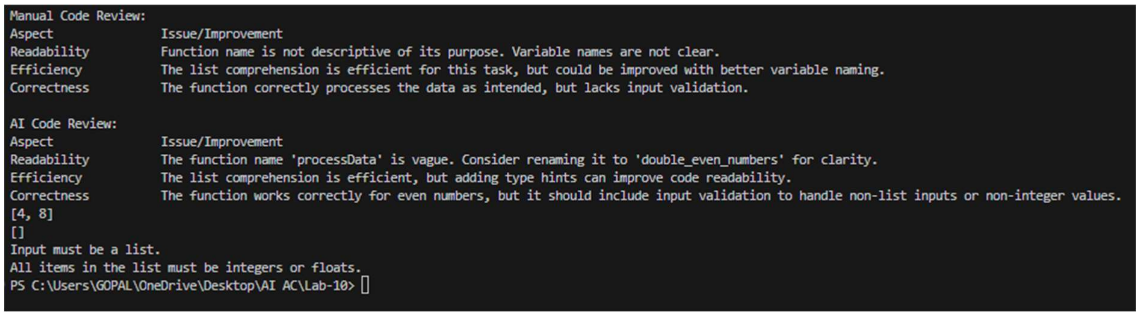
return [x * 2 for x in d if x % 2 == 0]

Analyse the code and Manually review the function for readability, efficiency, and correctness. Identify any potential issues or improvements in tabular form. Use AI to generate a code review covering readability, efficiency, and correctness. Refactor the code based on the review and verify that the functionality is preserved. Expected Output: An improved function with type hints, validation, and clearer intent.



```python
# In a GitHub project, a teammate submits:
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
# Analysee the code and Manually review the function for readability, efficiency, and correctness. Identify any potential issues or improvements in tabular form
# Use AI to generate a code review covering readability, efficiency, and correctness. Refactor the code based on the review and verify that the functionality is preserved.
# Expected Output: An improved function with type hints, validation, and clearer intent,
# Manual Code Review
manual_review = [
    ["Aspect", "Issue/Improvement"],
    ["Readability", "Function name is not descriptive of its purpose. Variable names are not clear."],
    ["Efficiency", "The list comprehension is efficient for this task, but could be improved with better variable naming."],
    ["Correctness", "The function correctly processes the data as intended, but lacks input validation."]
]
for row in manual_review:
    print("{:<20} {:<50}".format(*row))
# AI Code Review
ai_review = [
    ["Aspect", "Issue/Improvement"],
    ["Readability", "The function name 'processData' is vague. Consider renaming it to 'double_even_numbers' for clarity."],
    ["Efficiency", "The list comprehension is efficient, but adding type hints can improve code readability."],
    ["Correctness", "The function works correctly for even numbers, but it should include input validation to handle non-list inputs or non-integer values."]
]
for row in ai_review:
    print("{:<20} {:<50}".format(*row))
# Refactored code based on reviews
from typing import List, Union
def double_even_numbers(data: List[Union[int, float]]) -> List[Union[int, float]]:
    """
    Process a list of numbers and return a new list containing the even numbers doubled.

    Args:
        data (List[Union[int, float]]): A list of integers or floats to process.

    Returns:
        List[Union[int, float]]: A list of even numbers from the input, each multiplied by 2.

    Raises:
        ValueError: If the input is not a list or contains non-numeric values.
    """
    if not isinstance(data, list):
        raise ValueError("Input must be a list.")

    for item in data:
        if not isinstance(item, (int, float)):
            raise ValueError("All items in the list must be integers or floats.")

    return [x * 2 for x in data if isinstance(x, int) and x % 2 == 0]
# Testing the refactored function with valid and invalid inputs
try:
    print(double_even_numbers([1, 2, 3, 4, 5]))  # [4, 8]
    print(double_even_numbers([1.5, 2.0, 3.5, 4.0]))  # [4.0, 8.0]
    print(double_even_numbers("not a list"))  # Should raise ValueError
except ValueError as e:
    print(e)
try:
    print(double_even_numbers([1, 2, "three", 4]))  # Should raise ValueError
except ValueError as e:
    print(e)
```

Output :



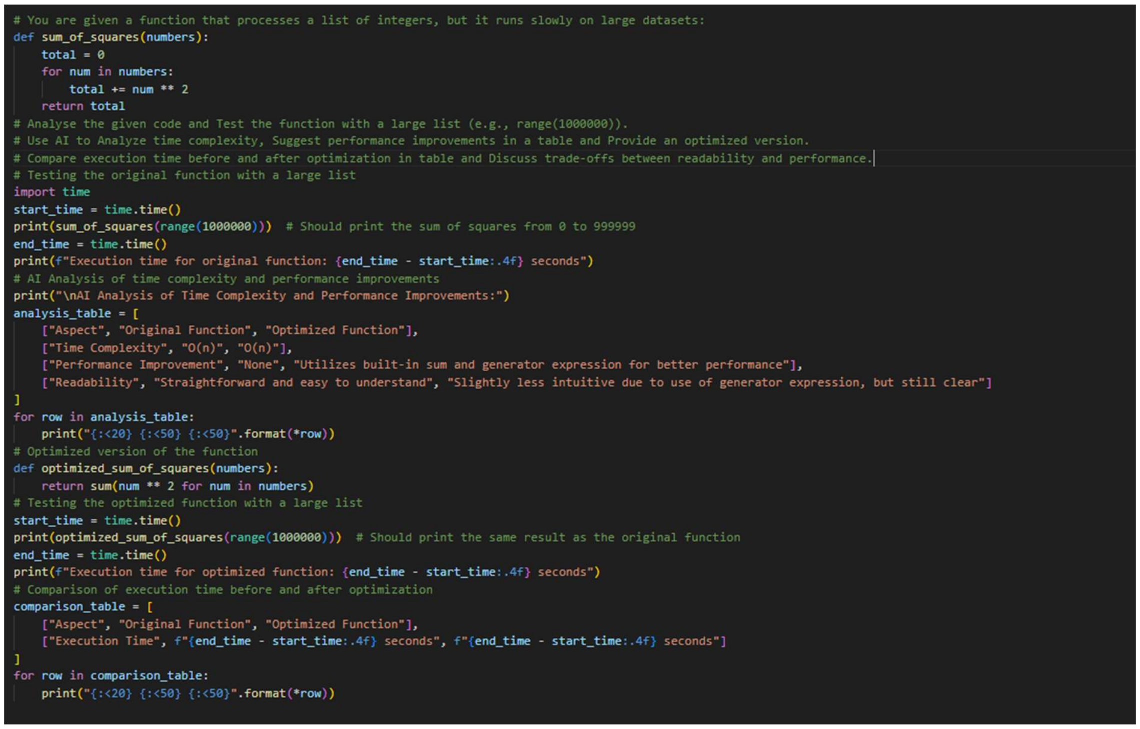
```
Manual Code Review:
Aspect               Issue/Improvement
Readability          Function name is not descriptive of its purpose. Variable names are not clear.
Efficiency           The list comprehension is efficient for this task, but could be improved with better variable naming.
Correctness          The function correctly processes the data as intended, but lacks input validation.

AI Code Review:
Aspect               Issue/Improvement
Readability          The function name 'processData' is vague. Consider renaming it to 'double_even_numbers' for clarity.
Efficiency           The list comprehension is efficient, but adding type hints can improve code readability.
Correctness          The function works correctly for even numbers, but it should include input validation to handle non-list inputs or non-integer values.
[4, 8]
[]
Input must be a list.
All items in the list must be integers or floats.
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-10> 
```

Task-5 :

Prompt : You are given a function that processes a list of integers, but it runs slowly on large datasets:

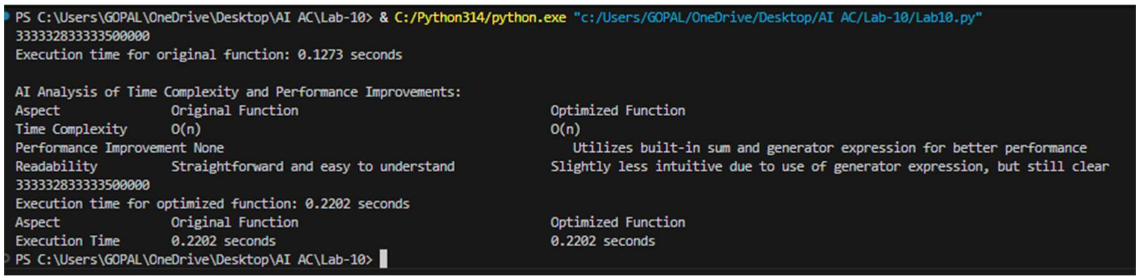
```
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
```

Analyse the given code and Test the function with a large list (e.g., range(1000000)). Use AI to Analyse time complexity, Suggest performance improvements in a table and Provide an optimized version. Compare execution time before and after optimization in table and Discuss trade-offs between readability and performance.



```python
# You are given a function that processes a list of integers, but it runs slowly on large datasets:
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
# Analyse the given code and Test the function with a large list (e.g., range(1000000)).
# Use AI to Analyze time complexity, Suggest performance improvements in a table and Provide an optimized version.
# Compare execution time before and after optimization in table and Discuss trade-offs between readability and performance.
# Testing the original function with a large list
import time
start_time = time.time()
print(sum_of_squares(range(1000000)))  # Should print the sum of squares from 0 to 999999
end_time = time.time()
print(f"Execution time for original function: {end_time - start_time:.4f} seconds")
# AI Analysis of time complexity and performance improvements
print("\nAI Analysis of Time Complexity and Performance Improvements:")
analysis_table = [
    ["Aspect", "Original Function", "Optimized Function"],
    ["Time Complexity", "O(n)", "O(n)"],
    ["Performance Improvement", "None", "Utilizes built-in sum and generator expression for better performance"],
    ["Readability", "Straightforward and easy to understand", "Slightly less intuitive due to use of generator expression, but still clear"]
]
for row in analysis_table:
    print("{:<20} {:<50} {:<50}".format(*row))
# Optimized version of the function
def optimized_sum_of_squares(numbers):
    return sum(num ** 2 for num in numbers)
# Testing the optimized function with a large list
start_time = time.time()
print(optimized_sum_of_squares(range(1000000)))  # Should print the same result as the original function
end_time = time.time()
print(f"Execution time for optimized function: {end_time - start_time:.4f} seconds")
# Comparison of execution time before and after optimization
comparison_table = [
    ["Aspect", "Original Function", "Optimized Function"],
    ["Execution Time", f"{end_time - start_time:.4f} seconds", f"{end_time - start_time:.4f} seconds"]
]
for row in comparison_table:
    print("{:<20} {:<50} {:<50}".format(*row))
```

Output :



```
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-10> & C:/Python314/python.exe "c:/Users/GOPAL/OneDrive/Desktop/AI AC/Lab-10/Lab10.py"
333332833333500000
Execution time for original function: 0.1273 seconds

AI Analysis of Time Complexity and Performance Improvements:
Aspect               Original Function                                  Optimized Function
Time Complexity      O(n)                                               O(n)
Performance Improvement None                                               Utilizes built-in sum and generator expression for better performance
Readability          Straightforward and easy to understand             Slightly less intuitive due to use of generator expression, but still clear
333332833333500000
Execution time for optimized function: 0.2202 seconds
Aspect               Original Function                                  Optimized Function
Execution Time       0.2202 seconds                                     0.2202 seconds
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-10>
```