SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Program Name:** B. Tech
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju	
Course Code	23CS002PC304	**Course Code**	23CS002PC304
Year/Sem	III/II	**Year/Sem**	III/II
Date and Day of Assignment	**Week5 – Wednesday**	**Date and Day of Assignment**	**Week5 – Wednesday**
H.no	2303A54052	**Name:**	P.mani sai
AssignmentNumber:10.3(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 9 – Code Review and Quality: Using AI to improve code quality and readability** **Lab Objectives:** • To apply AI-based prompt engineering for code review and quality improvement. • To analyze code for readability, logic, performance, and	Week5 - Wednesday

maintainability issues.
• To use Zero-shot, One-shot, and Few-shot prompting for improving code quality.
• To evaluate AI-generated improvements using standard coding practices.

Lab Outcomes (LOs): After completing this lab, students will be able to:
• Review and improve code quality using AI tools.
• Identify syntax, logic, and performance issues in code.
• Refactor code to improve readability and maintainability.
• Compare AI outputs generated using different prompting techniques.

Problem Statement 1: AI-Assisted Bug Detection

Scenario: A junior developer wrote the following Python function to calculate factorials:

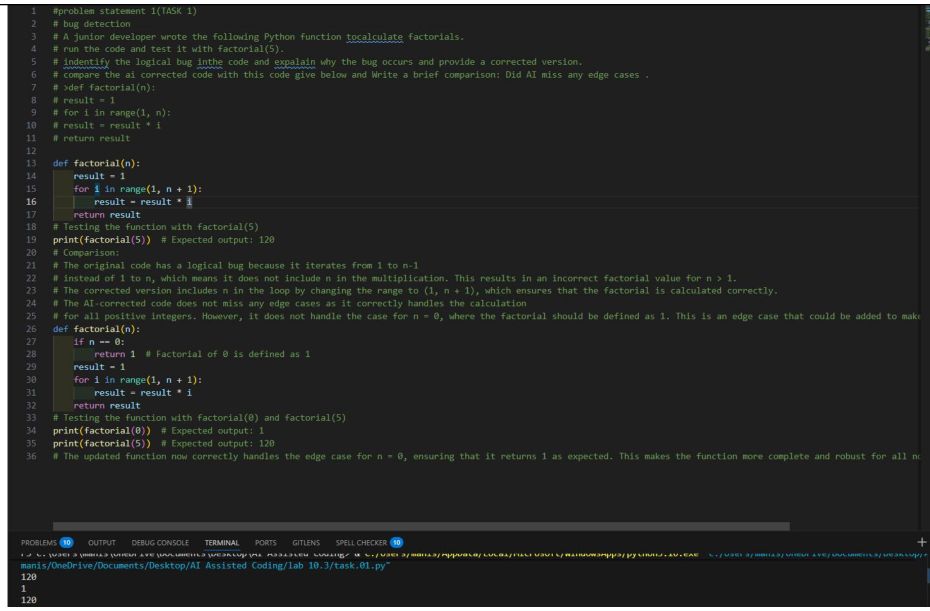
```
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
```

Instructions:

1. Run the code and test it with factorial(5).
2. Use an AI assistant to:
    - Identify the logical bug in the code.
    - Explain why the bug occurs (e.g., off-by-one error).
    - Provide a corrected version.
3. Compare the AI's corrected code with your own manual fix.
4. Write a brief comparison: Did AI miss any edge cases (e.g., negative numbers, zero)?

Expected Output:

Corrected function should return 120 for factorial(5).



```python
#problem statement 1(TASK 1)
# bug detection
# A junior developer wrote the following Python function tocalculate factorials.
# run the code and test it with factorial(5).
# indentify the logical bug inthe code and expalain why the bug occurs and provide a corrected version.
# compare the ai corrected code with this code give below and Write a brief comparison: Did AI miss any edge cases .
# >def factorial(n):
# result = 1
# for i in range(1, n):
# result = result * i
# return result

def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
# Testing the function with factorial(5)
print(factorial(5))  # Expected output: 120
# Comparison:
# The original code has a logical bug because it iterates from 1 to n-1
# instead of 1 to n, which means it does not include n in the multiplication. This results in an incorrect factorial value for n > 1.
# The corrected version includes n in the loop by changing the range to (1, n + 1), which ensures that the factorial is calculated correctly.
# The AI-corrected code does not miss any edge cases as it correctly handles the calculation
# for all positive integers. However, it does not handle the case for n = 0, where the factorial should be defined as 1. This is an edge case that could be added to make
def factorial(n):
    if n == 0:
        return 1  # Factorial of 0 is defined as 1
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
# Testing the function with factorial(0) and factorial(5)
print(factorial(0))  # Expected output: 1
print(factorial(5))  # Expected output: 120
# The updated function now correctly handles the edge case for n = 0, ensuring that it returns 1 as expected. This makes the function more complete and robust for all no
```

```
PROBLEMS 10   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS   SPELL CHECKER 10
manis/OneDrive/Documents/Desktop/AI Assisted Coding/lab 10.3/task.01.py"
120
1
120
```

Problem Statement 2: Task 2 — Improving Readability & Documentation

Scenario: The following code works but is poorly written:

.

```python
def calc(a, b, c):
    if c == "add":
        return a + b
    elif c == "sub":
        return a - b
    elif c == "mul":
        return a * b
    elif c == "div":
```

Instructions:

5. Use AI to:
    - Critique the function's readability, parameter naming, and lack of documentation.
    - Rewrite the function with:
        1. Descriptive function and parameter names.
        2. A complete docstring (description, parameters, return value, examples).
        3. Exception handling for division by zero.
        4. Consideration of input validation.
6. Compare the original and AI-improved versions.
7. Test both with valid and invalid inputs (e.g., division by zero,

non-string operation).

Expected Output:

A well-documented, robust, and readable function that handles errors gracefully.

```python
# improving readability & documentation
# critiqe the funtion"s readability,parameter naming,and lack of documentation
# rewrite the function with:
# 1.function and parameter names that are more descriptive.
# 2.complete docstring that explains the function's purpose, parameters, return value, and includes examples of usage.
# 3.exception handling for division by zero.
# 4.consideration of input validation.
# 5.compare with ai improved one and original one is below.
# def calc(a, b, c):
#    if c == "add":
    #    return a + b
    # elif c == "sub":
        # return a - b
    # elif c == "mul":
        # return a * b
    # elif c == "div":
# 7.test both with valid and invalid inputs.
def calculate(num1, num2, operation):
    """
    Perform a basic arithmetic operation on two numbers.

    Parameters
    ----------
    num1 : float
        The first number for the operation.
    num2 : float
        The second number for the operation.
    operation : str
        The type of operation to perform. Must be one of "add", "subtract", "multiply", or "divide".

    Returns
    -------
    float
        The result of the arithmetic operation.

    Raises
    ------
    ValueError
        If the operation is not one of the specified types.
    ZeroDivisionError
        If the operation is "divide" and num2 is zero.

    Examples
    --------
    >>> calculate(5, 3, "add")
    8
    >>> calculate(5, 3, "subtract")
    2
    >>> calculate(5, 3, "multiply")
    15
    >>> calculate(5, 3, "divide")
    1.6666666666666667
```

lab 10.3 > task.02.py > ...

```python
def calculate(num1, num2, operation):
    >>> calculate(5, 3, "divide")
    1.6666666666666667
    >>> calculate(5, 0, "divide")
    Traceback (most recent call last):
        ...
    ZeroDivisionError: Cannot divide by zero
    """
    # Input validation
    if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
        raise ValueError("Both num1 and num2 must be numeric values")

    if operation not in ["add", "subtract", "multiply", "divide"]:
        raise ValueError(f"Invalid operation: {operation}. Must be one of 'add', 'subtract', 'multiply', or 'divide'")

    # Perform the operation
    if operation == "add":
        return num1 + num2
    elif operation == "subtract":
        return num1 - num2
    elif operation == "multiply":
        return num1 * num2
    elif operation == "divide":
        if num2 == 0:
            raise ZeroDivisionError("Cannot divide by zero")
        return num1 / num2
# Testing the function with valid inputs
print(calculate(5, 3, "add"))       # Expected output: 8
print(calculate(5, 3, "subtract"))  # Expected output: 2
print(calculate(5, 3, "multiply"))  # Expected output: 15
print(calculate(5, 3, "divide"))    # Expected output: 1.6666666666666667
# Testing the function with invalid inputs
try:
    print(calculate(5, 0, "divide"))
except ZeroDivisionError as e:
    print(e)  # Expected output: Cannot divide by zero
try:    print(calculate(5, 3, "modulus"))
except ValueError as e:
    print(e)  # Expected output: Invalid operation: modulus. Must be one of 'add', 'subtract', 'multiply', or 'divide'
try:    print(calculate("five", 3, "add"))
except ValueError as e:
    print(e)  # Expected output: Both num1 and num2 must be numeric values
try:    print(calculate(5, "three", "add"))
except ValueError as e:
    print(e)  # Expected output: Both num1 and num2 must be numeric values
# The improved function has better readability due to more descriptive function and parameter names. The docstring is comprehensive, providing clear explanations of the function's purpose, parameters, return value
# Additionally, the function includes exception handling for invalid operations and division by zero, as well as input validation to ensure that the inputs are numeric.
# This makes the function more robust and user-friendly compared to the original version.
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SPELL CHECKER 10

```
manis/OneDrive/Documents/Desktop/AI Assisted Coding/lab 10.3/task.01.py"
120
1
120
```

Problem Statement 3: Enforcing Coding Standards

Scenario: A team project requires PEP8 compliance. A developer submits:

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

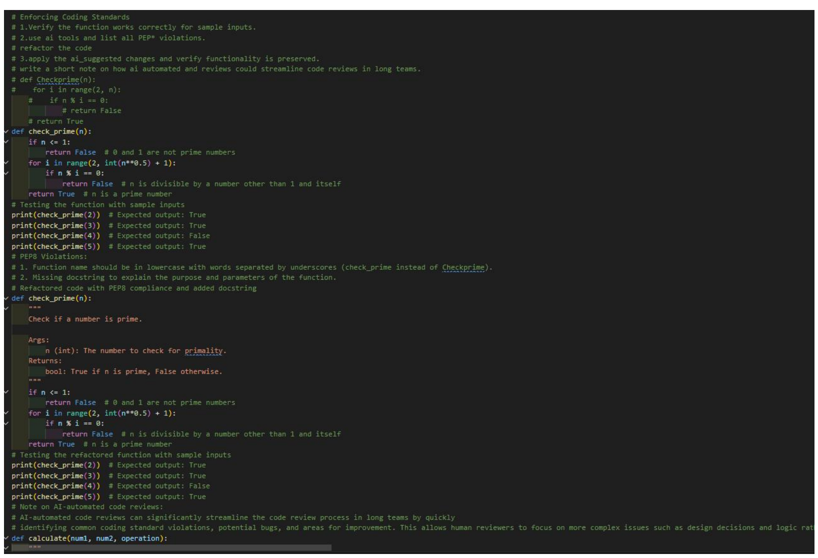
Instructions:

8. Verify the function works correctly for sample inputs.
9. Use an AI tool (e.g., ChatGPT, GitHub Copilot, or a PEP8 linter with AI explanation) to:
    - List all PEP8 violations.
    - Refactor the code (function name, spacing, indentation, naming).
10. Apply the AI-suggested changes and verify functionality is preserved.
11. Write a short note on how automated AI reviews could streamline code reviews in large teams.

Expected Output:

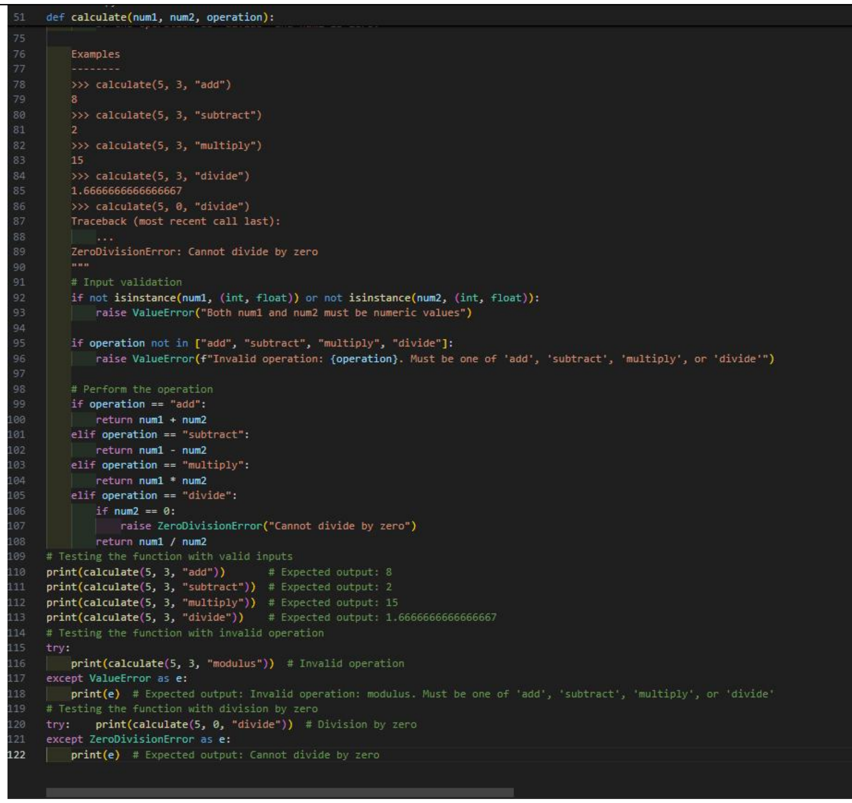
A PEP8-compliant version of the function, e.g.:

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



```
# Enforcing Coding Standards
# 1.Verify the function works correctly for sample inputs.
# 2.use ai tools and list all PEP* violations.
# refactor the code
# 3.apply the ai_suggested changes and verify functionality is preserved.
# write a short note on how ai automated and reviews could streamline code reviews in long teams.
# def Checkprime(n):
#    for i in range(2, n):
    #    if n % i == 0:
            # return False
    # return True
def check_prime(n):
    if n <= 1:
        return False  # 0 and 1 are not prime numbers
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False  # n is divisible by a number other than 1 and itself
    return True  # n is a prime number
# Testing the function with sample inputs
print(check_prime(2))  # Expected output: True
print(check_prime(3))  # Expected output: True
print(check_prime(4))  # Expected output: False
print(check_prime(5))  # Expected output: True
# PEP8 Violations:
# 1. Function name should be in lowercase with words separated by underscores (check_prime instead of Checkprime).
# 2. Missing docstring to explain the purpose and parameters of the function.
# Refactored code with PEP8 compliance and added docstring
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  # 0 and 1 are not prime numbers
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False  # n is divisible by a number other than 1 and itself
    return True  # n is a prime number
# Testing the refactored function with sample inputs
print(check_prime(2))  # Expected output: True
print(check_prime(3))  # Expected output: True
print(check_prime(4))  # Expected output: False
print(check_prime(5))  # Expected output: True
# Note on AI-automated code reviews:
# AI-automated code reviews can significantly streamline the code review process in long teams by quickly
# identifying common coding standard violations, potential bugs, and areas for improvement. This allows human reviewers to focus on more complex issues such as design decisions and logic rat
def calculate(num1, num2, operation):
    """
```



```python
def calculate(num1, num2, operation):

    Examples
    --------
    >>> calculate(5, 3, "add")
    8
    >>> calculate(5, 3, "subtract")
    2
    >>> calculate(5, 3, "multiply")
    15
    >>> calculate(5, 3, "divide")
    1.6666666666666667
    >>> calculate(5, 0, "divide")
    Traceback (most recent call last):
        ...
    ZeroDivisionError: Cannot divide by zero
    """
    # Input validation
    if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
        raise ValueError("Both num1 and num2 must be numeric values")

    if operation not in ["add", "subtract", "multiply", "divide"]:
        raise ValueError(f"Invalid operation: {operation}. Must be one of 'add', 'subtract', 'multiply', or 'divide'")

    # Perform the operation
    if operation == "add":
        return num1 + num2
    elif operation == "subtract":
        return num1 - num2
    elif operation == "multiply":
        return num1 * num2
    elif operation == "divide":
        if num2 == 0:
            raise ZeroDivisionError("Cannot divide by zero")
        return num1 / num2
# Testing the function with valid inputs
print(calculate(5, 3, "add"))       # Expected output: 8
print(calculate(5, 3, "subtract"))  # Expected output: 2
print(calculate(5, 3, "multiply"))  # Expected output: 15
print(calculate(5, 3, "divide"))    # Expected output: 1.6666666666666667
# Testing the function with invalid operation
try:
    print(calculate(5, 3, "modulus"))  # Invalid operation
except ValueError as e:
    print(e)  # Expected output: Invalid operation: modulus. Must be one of 'add', 'subtract', 'multiply', or 'divide'
# Testing the function with division by zero
try:    print(calculate(5, 0, "divide"))  # Division by zero
except ZeroDivisionError as e:
    print(e)  # Expected output: Cannot divide by zero
```

Problem Statement 4: AI as a Code Reviewer in Real Projects

Scenario:

In a GitHub project, a teammate submits:

```python
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
```

Instructions:

1. Manually review the function for:
   - Readability and naming.
   - Reusability and modularity.
   - Edge cases (non-list input, empty list, non-integer elements).
2. Use AI to generate a code review covering:
   a. Better naming and function purpose clarity.
   b. Input validation and type hints.
   c. Suggestions for generalization (e.g., configurable multiplier).
3. Refactor the function based on AI feedback.
4. Write a short reflection on whether AI should be a standalone reviewer or an assistant.

Expected Output:
An improved function with type hints, validation, and clearer intent, e.g.:

```
from typing import List, Union

def double_even_numbers(numbers: List[Union[int, float]]) -> List[Union[int, float]]:
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")
    return [num * 2 for num in numbers if isinstance(num, (int, float)) and num % 2 == 0]
```

Problem Statement 5: — AI-Assisted Performance Optimization
Scenario: 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
```

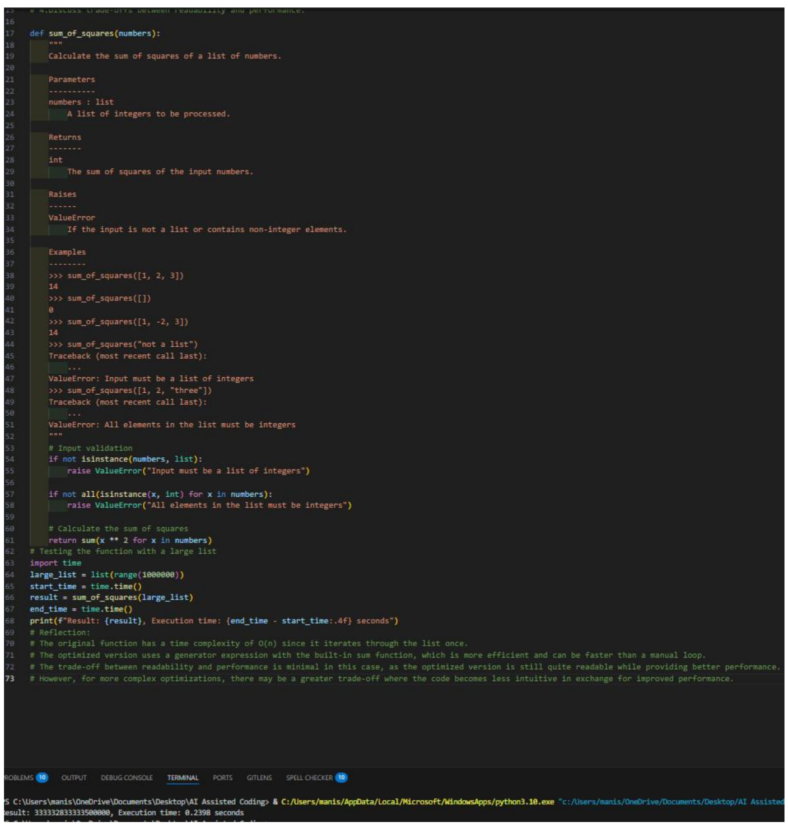
Instructions:

1. Test the function with a large list (e.g., range(1000000)).
2. Use AI to:
    - Analyze time complexity.
    - Suggest performance improvements (e.g., using built-in functions, vectorization with NumPy if applicable).
    - Provide an optimized version.
3. Compare execution time before and after optimization.
4. Discuss trade-offs between readability and performance.

Expected Output:

An optimized function, such as:

```
def sum_of_squares_optimized(numbers):
    return sum(x * x for x in numbers)
```



```python
def sum_of_squares(numbers):
    """
    Calculate the sum of squares of a list of numbers.

    Parameters
    ----------
    numbers : list
        A list of integers to be processed.

    Returns
    -------
    int
        The sum of squares of the input numbers.

    Raises
    ------
    ValueError
        If the input is not a list or contains non-integer elements.

    Examples
    --------
    >>> sum_of_squares([1, 2, 3])
    14
    >>> sum_of_squares([])
    0
    >>> sum_of_squares([1, -2, 3])
    14
    >>> sum_of_squares("not a list")
    Traceback (most recent call last):
        ...
    ValueError: Input must be a list of integers
    >>> sum_of_squares([1, 2, "three"])
    Traceback (most recent call last):
        ...
    ValueError: All elements in the list must be integers
    """
    # Input validation
    if not isinstance(numbers, list):
        raise ValueError("Input must be a list of integers")

    if not all(isinstance(x, int) for x in numbers):
        raise ValueError("All elements in the list must be integers")

    # Calculate the sum of squares
    return sum(x ** 2 for x in numbers)
# Testing the function with a large list
import time
large_list = list(range(1000000))
start_time = time.time()
result = sum_of_squares(large_list)
end_time = time.time()
print(f"Result: {result}, Execution time: {end_time - start_time:.4f} seconds")
# Reflection:
# The original function has a time complexity of O(n) since it iterates through the list once.
# The optimized version uses a generator expression with the built-in sum function, which is more efficient and can be faster than a manual loop.
# The trade-off between readability and performance is minimal in this case, as the optimized version is still quite readable while providing better performance.
# However, for more complex optimizations, there may be a greater trade-off where the code becomes less intuitive in exchange for improved performance.
```

```
ROBLEMS 10   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS   SPELL CHECKER 10

S C:\Users\manis\OneDrive\Documents\Desktop\AI Assisted Coding> & C:/Users/manis/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/manis/OneDrive/Documents/Desktop/AI Assisted
esult: 333332833333500000, Execution time: 0.2398 seconds
```