

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: <b>B. Tech</b>		Assignment Type: Lab	Program Name: <b>B. Tech</b>
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: <b>10.3(Present assignment number)/24(Total number of assignments)</b>			
Q.No.	Question	Expected Time to complete	
1	<b>Lab 9 – Code Review and Quality: Using AI to improve code quality and readability</b>  <b>Lab Objectives:</b> <ul style="list-style-type: none"> <li>• To apply AI-based prompt engineering for code review and quality improvement.</li> <li>• To analyze code for readability, logic, performance, and</li> </ul>	Week5 - Wednesday	

	<p>maintainability issues.</p> <ul style="list-style-type: none"> <li>• To use Zero-shot, One-shot, and Few-shot prompting for improving code quality.</li> <li>• To evaluate AI-generated improvements using standard coding practices.</li> </ul> <p><b>Lab Outcomes (LOs):</b> After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> <li>• Review and improve code quality using AI tools.</li> <li>• Identify syntax, logic, and performance issues in code.</li> <li>• Refactor code to improve readability and maintainability.</li> <li>• Compare AI outputs generated using different prompting techniques.</li> </ul> <p><b>Problem Statement 1: AI-Assisted Bug Detection</b></p> <p><b>Scenario:</b> A junior developer wrote the following Python function to calculate factorials:</p> <pre>def factorial(n):     result = 1     for i in range(1, n):         result = result * i     return result</pre> <p><b>Instructions:</b></p> <ol style="list-style-type: none"> <li>1. Run the code and test it with factorial(5).</li> <li>2. Use an AI assistant to:             <ul style="list-style-type: none"> <li>○ Identify the logical bug in the code.</li> <li>○ Explain why the bug occurs (e.g., off-by-one error).</li> <li>○ Provide a corrected version.</li> </ul> </li> <li>3. Compare the AI's corrected code with your own manual fix.</li> <li>4. Write a brief comparison: Did AI miss any edge cases (e.g., negative numbers, zero)?</li> </ol> <p><b>Expected Output:</b> Corrected function should return 120 for factorial(5).</p>	
--	--	--

```

1 #problem statement 1(TASK 1)
2 # bug detection
3 # A junior developer wrote the following Python function to calculate factorials.
4 # run the code and test it with factorial(5).
5 # Identify the logical bug in the code and explain why the bug occurs and provide a corrected version.
6 # compare the ai corrected code with this code give below and Write a brief comparison: Did AI miss any edge cases .
7 >def factorial(n):
8     # result = 1
9     for i in range(1, n):
10        # result = result * i
11    # return result
12
13
14    def factorial(n):
15        result = 1
16        for i in range(1, n + 1):
17            result = result * i
18        return result
19    # Testing the function with factorial(5)
20    print(factorial(5)) # Expected output: 120
21
22    # Comparison:
23    # The original code has a logical bug because it iterates from 1 to n-1
24    # 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.
25    # The corrected version includes n in the loop by changing the range to (1, n + 1), which ensures that the factorial is calculated correctly.
26    # The AI-corrected code does not miss any edge cases as it correctly handles the calculation
27    # 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
28
29    def factorial(n):
30        if n == 0:
31            return 1 # Factorial of 0 is defined as 1
32        result = 1
33        for i in range(1, n + 1):
34            result = result * i
35        return result
36    # Testing the function with factorial(0) and factorial(5)
37    print(factorial(0)) # Expected output: 1
38    print(factorial(5)) # Expected output: 120
39    # 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 n

```

## Problem Statement 2: Task 2 — Improving Readability & Documentation

**Scenario:**The following code works but is poorly written:

```

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.

```
# Improving readability & documentation
# rewrite the function'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 an 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":
    #     return a / b

# 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
    """
```

```
#!/usr/bin/env python3
10 def calculate(num1, num2, operation):
11     """
12     Perform a basic arithmetic operation on two numbers.
13     """
14     # Input validation
15     if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
16         raise ValueError("Both num1 and num2 must be numeric values")
17     if operation not in ["add", "subtract", "multiply", "divide"]:
18         raise ValueError(f"Invalid operation: {operation}. Must be one of 'add', 'subtract', 'multiply', or 'divide'")
19
20     # Perform the operation
21     if operation == "add":
22         return num1 + num2
23     elif operation == "subtract":
24         return num1 - num2
25     elif operation == "multiply":
26         return num1 * num2
27     elif operation == "divide":
28         if num2 == 0:
29             raise ZeroDivisionError("Cannot divide by zero")
30         return num1 / num2
31
32 # Testing the function with valid inputs
33 print(calculate(5, 3, "add")) # Expected output: 8
34 print(calculate(5, 3, "subtract")) # Expected output: 2
35 print(calculate(5, 3, "multiply")) # Expected output: 15
36 print(calculate(5, 3, "divide")) # Expected output: 1.6666666666666667
37
38 # Testing the function with invalid inputs
39 try:
40     print(calculate(5, 0, "divide"))
41 except ZeroDivisionError as e:
42     print(f"Error: {e}")
43
44 try:
45     print(calculate(5, 3, "modulus"))
46 except ValueError as e:
47     print(f"Error: {e}")
48
49 try:
50     print(calculate("five", 3, "add"))
51 except ValueError as e:
52     print(f"Error: {e}")
53
54 try:
55     print(calculate(5, "three", "mul"))
56 except ValueError as e:
57     print(f"Error: {e}")
58
59 # 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
60 # 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.
61 # This makes the function more robust and user-friendly compared to the original version.
62
63 if __name__ == "__main__":
64     # Example usage
65     result = calculate(10, 5, "add")
66     print(f"10 + 5 = {result}")
67
68     result = calculate(10, 5, "subtract")
69     print(f"10 - 5 = {result}")
70
71     result = calculate(10, 5, "multiply")
72     print(f"10 * 5 = {result}")
73
74     result = calculate(10, 5, "divide")
75     print(f"10 / 5 = {result}")
76
77     # Invalid operation
78     try:
79         result = calculate(10, 5, "modulus")
80     except ValueError as e:
81         print(f"Invalid operation: {e}")
82
83     # Invalid input
84     try:
85         result = calculate("ten", 5, "add")
86     except ValueError as e:
87         print(f"Invalid input: {e}")
88
89     # Division by zero
90     try:
91         result = calculate(10, 0, "divide")
92     except ZeroDivisionError as e:
93         print(f"Division by zero: {e}")
94
95     # Invalid operation and input
96     try:
97         result = calculate("ten", 0, "divide")
98     except (ValueError, ZeroDivisionError) as e:
99         print(f"Invalid operation and input: {e}")
100
101     # Test with floats
102     result = calculate(1.5, 2.5, "add")
103     print(f"1.5 + 2.5 = {result}")
104
105     result = calculate(1.5, 2.5, "multiply")
106     print(f"1.5 * 2.5 = {result}")
107
108     result = calculate(1.5, 2.5, "divide")
109     print(f"1.5 / 2.5 = {result}")
110
111     # Test with negative numbers
112     result = calculate(-5, 3, "add")
113     print(f"-5 + 3 = {result}")
114
115     result = calculate(-5, 3, "subtract")
116     print(f"-5 - 3 = {result}")
117
118     result = calculate(-5, 3, "multiply")
119     print(f"-5 * 3 = {result}")
120
121     result = calculate(-5, 3, "divide")
122     print(f"-5 / 3 = {result}")
123
124     # Test with zero
125     result = calculate(0, 5, "add")
126     print(f"0 + 5 = {result}")
127
128     result = calculate(0, 5, "subtract")
129     print(f"0 - 5 = {result}")
130
131     result = calculate(0, 5, "multiply")
132     print(f"0 * 5 = {result}")
133
134     result = calculate(0, 5, "divide")
135     print(f"0 / 5 = {result}")
136
137     # Test with large numbers
138     result = calculate(1000000, 1000000, "add")
139     print(f"1000000 + 1000000 = {result}")
140
141     result = calculate(1000000, 1000000, "multiply")
142     print(f"1000000 * 1000000 = {result}")
143
144     result = calculate(1000000, 1000000, "divide")
145     print(f"1000000 / 1000000 = {result}")
146
147     # Test with small numbers
148     result = calculate(0.001, 0.001, "add")
149     print(f"0.001 + 0.001 = {result}")
150
151     result = calculate(0.001, 0.001, "multiply")
152     print(f"0.001 * 0.001 = {result}")
153
154     result = calculate(0.001, 0.001, "divide")
155     print(f"0.001 / 0.001 = {result}")
156
157     # Test with mixed types
158     result = calculate(5, 3, "add")
159     print(f"5 + 3 = {result}")
160
161     result = calculate(5, 3, "subtract")
162     print(f"5 - 3 = {result}")
163
164     result = calculate(5, 3, "multiply")
165     print(f"5 * 3 = {result}")
166
167     result = calculate(5, 3, "divide")
168     print(f"5 / 3 = {result}")
169
170     # Test with strings
171     try:
172         result = calculate("5", "3", "add")
173     except ValueError as e:
174         print(f"Invalid input: {e}")
175
176     try:
177         result = calculate("5", "3", "subtract")
178     except ValueError as e:
179         print(f"Invalid input: {e}")
180
170
```

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:
  - o List all PEP8 violations.
  - o 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 PEP8 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 re
def calculate(nval, num1, operation):
```

```

51 def calculate(num1, num2, operation):
75
76     Examples
77     -----
78     >>> calculate(5, 3, "add")
79     8
80     >>> calculate(5, 3, "subtract")
81     2
82     >>> calculate(5, 3, "multiply")
83     15
84     >>> calculate(5, 3, "divide")
85     1.6666666666666667
86     >>> calculate(5, 0, "divide")
87     Traceback (most recent call last):
88     ...
89     ZeroDivisionError: Cannot divide by zero
90     """
91     # Input validation
92     if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
93         raise ValueError("Both num1 and num2 must be numeric values")
94
95     if operation not in ["add", "subtract", "multiply", "divide"]:
96         raise ValueError(f"Invalid operation: {operation}. Must be one of 'add', 'subtract', 'multiply', or 'divide'")
97
98     # Perform the operation
99     if operation == "add":
100         return num1 + num2
101     elif operation == "subtract":
102         return num1 - num2
103     elif operation == "multiply":
104         return num1 * num2
105     elif operation == "divide":
106         if num2 == 0:
107             raise ZeroDivisionError("Cannot divide by zero")
108         return num1 / num2
109
110     # Testing the function with valid inputs
111     print(calculate(5, 3, "add")) # Expected output: 8
112     print(calculate(5, 3, "subtract")) # Expected output: 2
113     print(calculate(5, 3, "multiply")) # Expected output: 15
114     print(calculate(5, 3, "divide")) # Expected output: 1.6666666666666667
115
116     # Testing the function with invalid operation
117     try:
118         print(calculate(5, 3, "modulus")) # Invalid operation
119     except ValueError as e:
120         print(e) # Expected output: Invalid operation: modulus. Must be one of 'add', 'subtract', 'multiply', or 'divide'
121
122     # Testing the function with division by zero
123     try:
124         print(calculate(5, 0, "divide")) # Division by zero
125     except ZeroDivisionError as e:
126         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:

```

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

```

### Instructions:

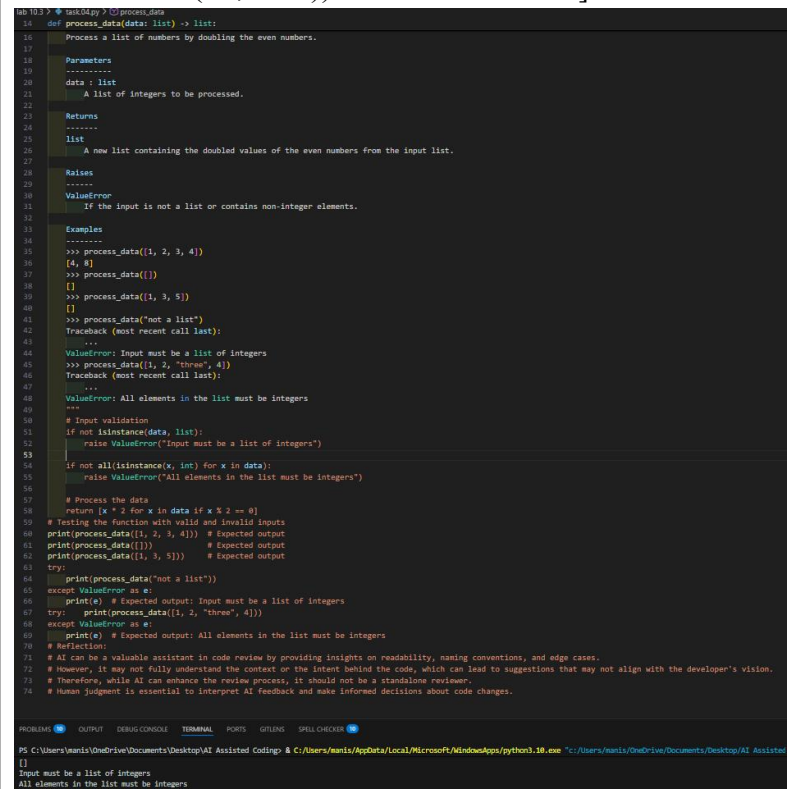
1. Manually review the function for:
  - Readability and naming.
  - Reusability and modularity.
  - Edge cases (non-list input, empty list, non-integer elements).
2. Use AI to generate a code review covering:
  - a. Better naming and function purpose clarity.
  - b. Input validation and type hints.
  - 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)
```

```
# This code is for educational purposes only and is not intended for production use.  
16  
17 def sum_of_squares(numbers):  
18     """  
19     Calculate the sum of squares of a list of numbers.  
20  
21     Parameters  
22     -----  
23     numbers : list  
24         A list of integers to be processed.  
25  
26     Returns  
27     -----  
28     int  
29         The sum of squares of the input numbers.  
30  
31     Raises  
32     -----  
33     ValueError  
34         If the input is not a list or contains non-integer elements.  
35  
36     Examples  
37     -----  
38     >>> sum_of_squares([1, 2, 3])  
39     14  
40     >>> sum_of_squares([])  
41     0  
42     >>> sum_of_squares([1, -2, 3])  
43     14  
44     >>> sum_of_squares("not a list")  
45     Traceback (most recent call last):  
46     ValueError: Input must be a list of integers  
47     >>> sum_of_squares([1, 2, "three"])  
48     Traceback (most recent call last):  
49     ValueError: All elements in the list must be integers  
50     """  
51  
52     # Input validation  
53     if not isinstance(numbers, list):  
54         raise ValueError("Input must be a list of integers")  
55  
56     if not all(isinstance(x, int) for x in numbers):  
57         raise ValueError("All elements in the list must be integers")  
58  
59     # Calculate the sum of squares  
60     return sum(x ** 2 for x in numbers)  
61  
62 # Testing the function with a large list  
63 import time  
64 large_list = list(range(1000000))  
65 start_time = time.time()  
66 result = sum_of_squares(large_list)  
67 end_time = time.time()  
68 print(f"Result: {result}, Execution time: {end_time - start_time:.4f} seconds")  
69  
70 # Reflection:  
71 # The original function has a time complexity of O(n) since it iterates through the list once.  
72 # 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.  
73 # The trade-off between readability and performance is minimal in this case, as the optimized version is still quite readable while providing better performance.  
74 # However, for more complex optimizations, there may be a greater trade-off where the code becomes less intuitive in exchange for improved performance.
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS SPELL CHECKER

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 Coding\sum\_of\_squares\_optimized.py"  
Result: 333120311500000, Execution time: 0.298 seconds