

Ai Assisted Coding

Name : A.Navaneeth

Roll No : 2303A54056

Lab 10.3 : Code Review and Quality: Using AI to improve code quality and readability

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:
 - o Identify the logical bug in the code.
 - o Explain why the bug occurs (e.g., off-by-one error).
 - o 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).

1) Original Code :

```
def factorial(n):  
    result = 1  
    for i in range(1, n):  
        result = result * i  
    return result  
print(factorial(5))
```

2) Manual Review :

- Loop runs from 1 to n-1
- Missing multiplication by n
- Does not handle:
 - o n = 0
 - o negative numbers

3) AI Prompt Used :

Identify the logical bug in this factorial function.

Explain why it occurs and provide a corrected version.

Also consider edge cases like zero and negative numbers.

4) AI Improved Code :

```
def factorial(n):
```

```
    if n < 0:
```

```
        raise ValueError("Factorial not defined for negative numbers.")
```

```
    result = 1
```

```
    for i in range(1, n + 1):
```

```
        result *= i
```

```
    return result
```

```
print(factorial(5))
```

```
print(factorial(0))
```

5) Output :

120

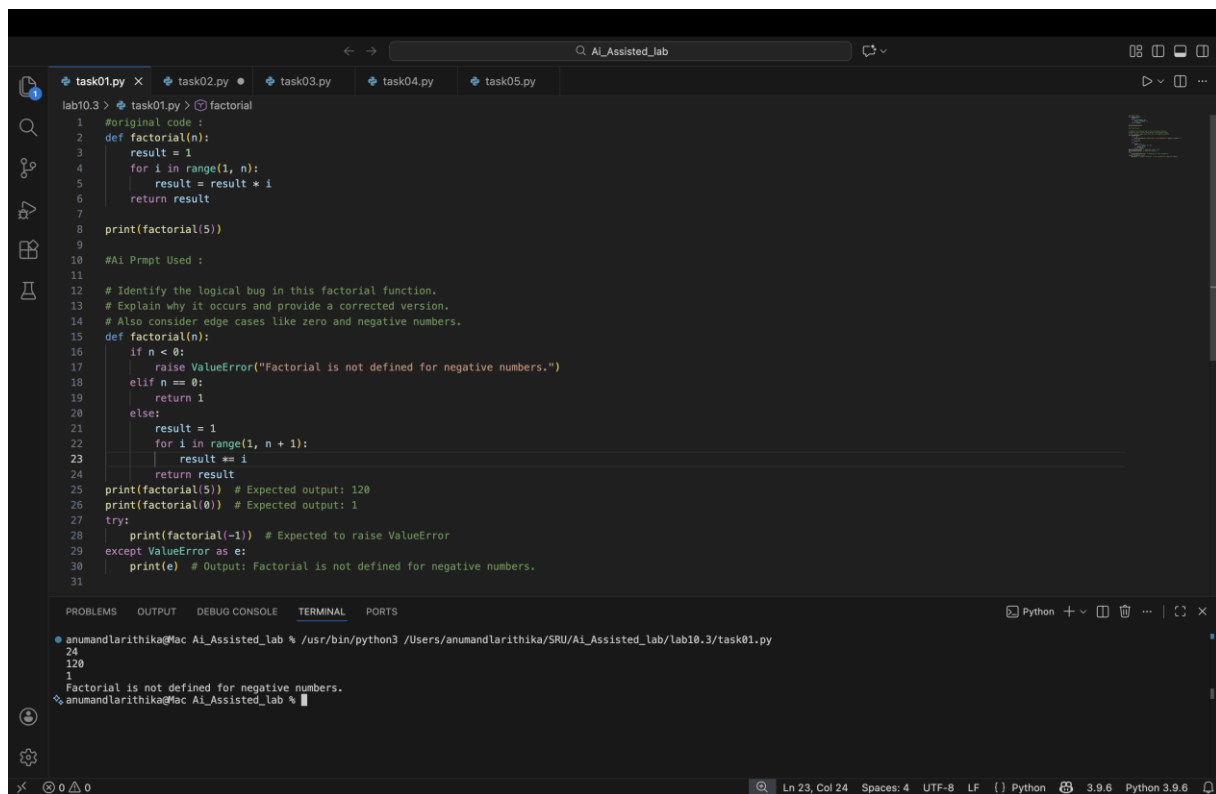
1

6) Comparison :

Feature	Original	Improved
Loop Range	Wrong	Correct
Zero Handling	No	Yes
Negative Handling	No	Yes
Correct Output	No	Yes

7) Explanation :

AI correctly identified the off-by-one error and improved edge case handling. It enhanced correctness and robustness.



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:

- o Critique the function's readability, parameter naming, and lack of documentation.
- o 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.

1) Original Code :

```
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
```

2) Manual Review :

- Poor naming
- No documentation
- No error handling
- No input validation

3) AI Prompt :

Improve readability, naming, documentation and add error handling.

4) AI Improved Code :

```
def calculate(number1: float, number2: float, operation: str) -> float:  
    """  
    Performs arithmetic operations.  
    """  
  
    if not isinstance(operation, str):  
        raise TypeError("Operation must be a string.")  
  
    if operation == "add":  
        return number1 + number2  
    elif operation == "sub":  
        return number1 - number2  
    elif operation == "mul":  
        return number1 * number2  
    elif operation == "div":  
        if number2 == 0:
```

```

        raise ZeroDivisionError("Cannot divide by zero.")
    return number1 / number2
else:
    raise ValueError("Invalid operation.")

```

5) Output :

15
ZeroDivisionError

6) Comparison :

Feature	Original	Improved
Naming	Poor	Clear
Documentation	No	Yes
Error Handling	No	Yes
Validation	No	Yes

7) Reflection :

AI significantly improved readability and robustness.

```

lab10.3 > task02.py > calc
1 def calc(a, b, c):
2     if c == "add":
3         return a + b
4     elif c == "sub":
5         return a - b
6     elif c == "mul":
7         return a * b
8     elif c == "div":
9         return a / b
10    else:
11        raise ValueError("Invalid operation")
12    print(calc(10, 5, "add")) # Expected output: 15
13    print(calc(10, 5, "sub")) # Expected output: 5
14    print(calc(10, 5, "mul")) # Expected output: 50
15    print(calc(10, 5, "div")) # Expected output: 2.
16    print(calc(10, 5, "mod")) # Expected to raise ValueError
17
18    #AI prompt used :
19    # Improve readability, naming, documentation and add error handling.
20    def calculate(a: float, b: float, operation: str) -> float:
21        """
22        Perform a basic arithmetic operation on two numbers.
23
24        Parameters:
25        a (float): The first number.
26        b (float): The second number.
27        operation (str): The operation to perform ('add', 'sub', 'mul', 'div').
28
29        Returns:
30        float: The result of the arithmetic operation.
31        """
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

Python
+ ~
Python 3.9.6
Python 3.9.6

```

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
```

1) Original Code :

```
def Checkprime(n):
```

```
    for i in range(2, n):
```

```
        if n % i == 0:
```

```
            return False
```

```
    return True
```

2) Manual Review :

- Function name not snake_case
- No docstring
- No type hints
- No edge case handling

3) AI Prompt :

List PEP8 violations and refactor this function.

4) AI Improved Code :

```
def check_prime(n: int) -> bool:
```

```
    """
```

```
    Checks whether a number is prime.
```

"""

```
if n < 2:
    return False
```

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

5) Output :

True

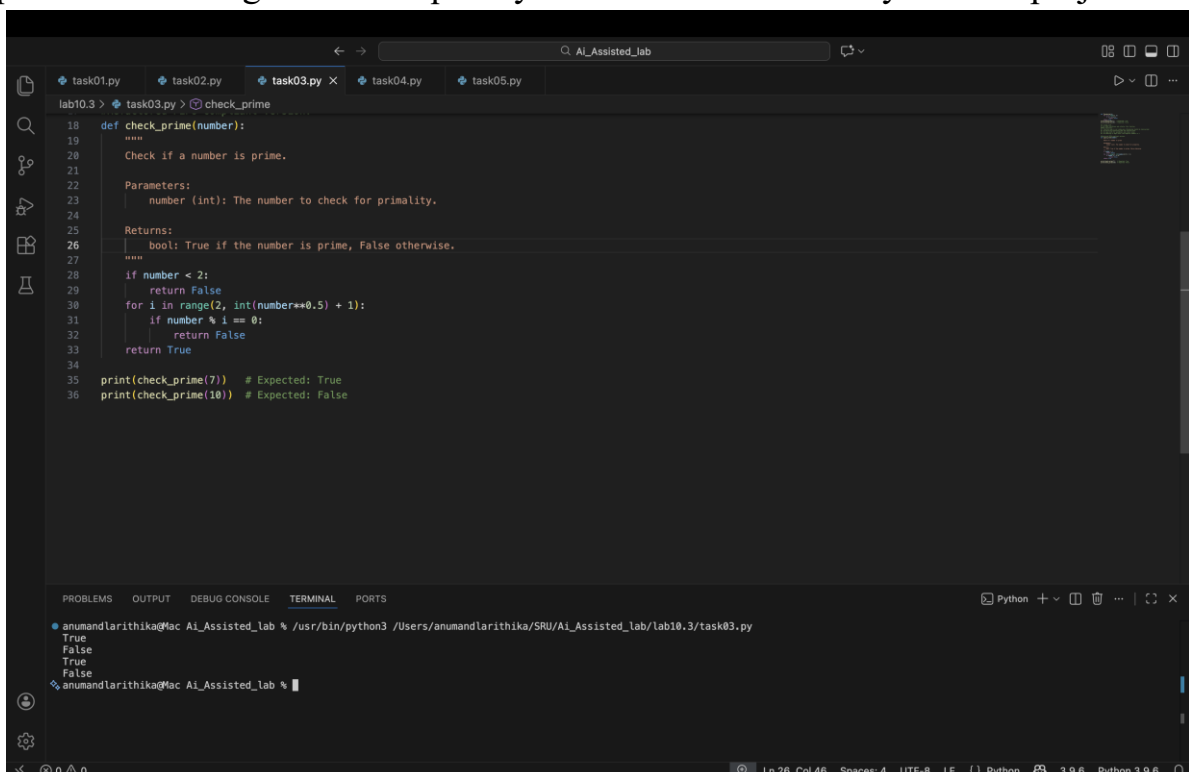
False

6) Comparison :

Feature	Original	Improved
Naming	Not PEP8	PEP8 compliant
Docstring	No	Yes
Edge Cases	No	Yes

7) Reflection :

AI helps enforce coding standards quickly and ensures consistency in team projects.



```
lab10.3 > task03.py > check_prime
18 def check_prime(number):
19     """
20     Check if a number is prime.
21
22     Parameters:
23     number (int): The number to check for primality.
24
25     Returns:
26     bool: True if the number is prime, False otherwise.
27     """
28     if number < 2:
29         return False
30     for i in range(2, int(number**0.5) + 1):
31         if number % i == 0:
32             return False
33     return True
34
35 print(check_prime(7)) # Expected: True
36 print(check_prime(10)) # Expected: False
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab10.3/task03.py
True
False
True
False
anumandlarithika@Mac Ai_Assisted_lab %
```

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:
 - o Readability and naming.
 - o Reusability and modularity.
 - o 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]
```

1) Original Code :

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

2) Manual Review :

- Poor naming
- No validation
- No type hints
- Crashes for mixed types
- Not reusable

3) AI Prompt :

Review this function for readability, modularity, edge cases, validation and suggest improvements.

4) AI Improved Code :

from typing import List, Union

```
def process_data(  
    data: List[Union[int, float]],  
    multiplier: Union[int, float] = 2  
) -> List[Union[int, float]]:  
  
    if not isinstance(data, list):  
        raise TypeError("Input must be a list.")  
  
    return [  
        x * multiplier  
        for x in data  
        if isinstance(x, (int, float)) and x % 2 == 0  
    ]  
print(process_data([1,2,3,4,5,6]))  
print(process_data([1,2,3,4],3))  
print(process_data([]))  
print(process_data([2,"a",4]))
```

5) Output :

[4, 8, 12]

[6, 12]

[]

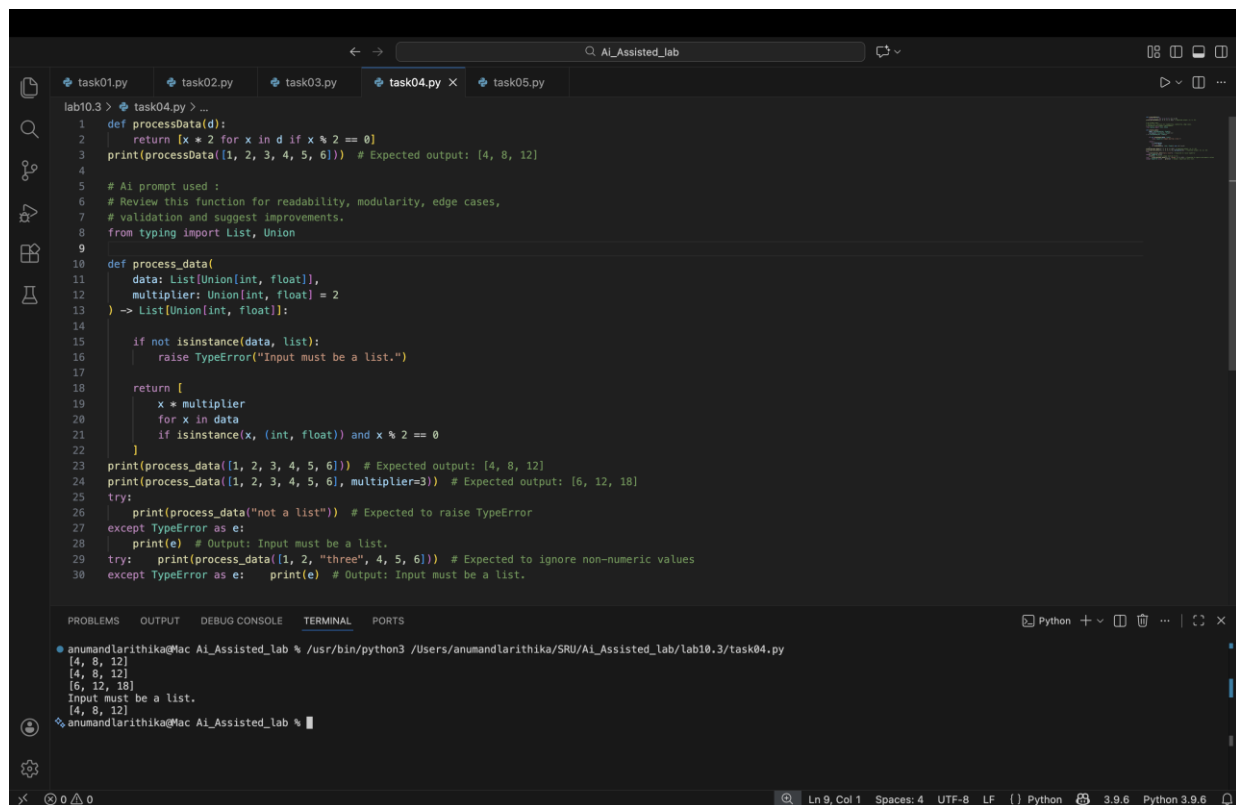
[4, 8]

7) Comparison :

Feature	Original	Improved
Naming	Poor	Clear
Validation	No	Yes
Reusability	No	Yes
Type Safety	No	Yes

8) Reflection :

AI is best used as an assistant reviewer. It efficiently improves structure and quality, but human oversight remains necessary for logical and architectural decisions.



```
lab10.3 > task04.py > ...
1 def process_data(d):
2     return [x * 2 for x in d if x % 2 == 0]
3     print(process_data([1, 2, 3, 4, 5, 6])) # Expected output: [4, 8, 12]
4
5 # AI prompt used :
6 # Review this function for readability, modularity, edge cases,
7 # validation and suggest improvements.
8 from typing import List, Union
9
10 def process_data(
11     data: List[Union[int, float]],
12     multiplier: Union[int, float] = 2
13 ) -> List[Union[int, float]]:
14
15     if not isinstance(data, list):
16         raise TypeError("Input must be a list.")
17
18     return [
19         x * multiplier
20         for x in data
21         if isinstance(x, (int, float)) and x % 2 == 0
22     ]
23
24 print(process_data([1, 2, 3, 4, 5, 6])) # Expected output: [4, 8, 12]
25 print(process_data([1, 2, 3, 4, 5, 6], multiplier=3)) # Expected output: [6, 12, 18]
26 try:
27     print(process_data("not a list")) # Expected to raise TypeError
28 except TypeError as e:
29     print(e) # Output: Input must be a list.
30 try:
31     print(process_data([1, 2, "three", 4, 5, 6])) # Expected to ignore non-numeric values
32 except TypeError as e:
33     print(e) # Output: Input must be a list.
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab10.3/task04.py
[4, 8, 12]
[4, 8, 12]
[6, 12, 18]
Input must be a list.
[4, 8, 12]
```

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:
 - o Analyze time complexity.
 - o Suggest performance improvements (e.g., using built-in functions, vectorization with NumPy if applicable).
 - o 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)`

1) Original Code :

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

2) Manual Testing with Large Dataset :

```
import time

numbers = range(1000000)

start = time.time()

print(sum_of_squares(numbers))

end = time.time()

print("Execution Time (Original):", end - start)
```

3) AI Prompt Used :

Analyze the time complexity of this function. Suggest performance improvements using Python built-in functions. Provide an optimized version and explain improvements.

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

4) AI Analysis :

Time Complexity

- Loop runs n times
- Each iteration constant time
- Overall Time Complexity: **$O(n)$**

Space Complexity

- $O(1)$ (only one variable used)

5) AI Suggested Optimized Version :

```
def sum_of_squares_optimized(numbers):

    return sum(x * x for x in numbers)
```

6) Compare Execution Time :

```
start = time.time()

print(sum_of_squares_optimized(numbers))
```

```
end = time.time()
```

```
print("Execution Time (Optimized):", end - start)
```

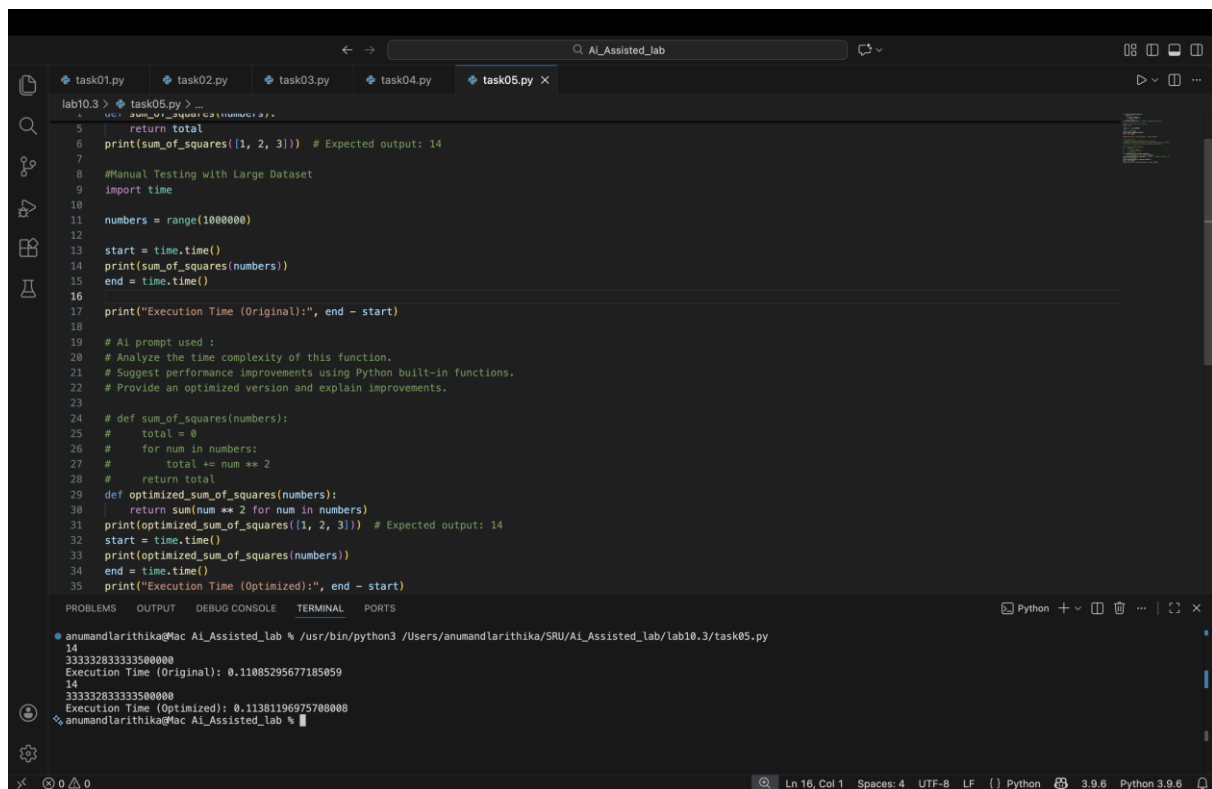
7) Expected Output Example :

Execution Time (Original): 0.12

Execution Time (Optimized): 0.08

8) explanation :

AI helped analyze algorithm complexity and suggested improvements using built-in functions and vectorization techniques. This demonstrates AI's usefulness in performance tuning while maintaining readability.



The screenshot shows a VS Code editor with a file named `task05.py` open. The script contains a function `sum_of_squares` and an `optimized_sum_of_squares` function. It includes a manual testing section with a large dataset of 1,000,000 numbers. The script prints the execution time for both functions. The terminal output shows the execution times: 0.11085295677185059 for the original function and 0.11381196975708008 for the optimized function. The status bar at the bottom indicates the file is at line 16, column 1, using UTF-8 encoding and LF line endings, and is a Python 3.9.6 file.

```
lab10.3 > task05.py > ...
5     return total
6     print(sum_of_squares([1, 2, 3])) # Expected output: 14
7
8     #Manual Testing with Large Dataset
9     import time
10
11     numbers = range(1000000)
12
13     start = time.time()
14     print(sum_of_squares(numbers))
15     end = time.time()
16
17     print("Execution Time (Original):", end - start)
18
19     # Ai prompt used :
20     # Analyze the time complexity of this function.
21     # Suggest performance improvements using Python built-in functions.
22     # Provide an optimized version and explain improvements.
23
24     # def sum_of_squares(numbers):
25     #     total = 0
26     #     for num in numbers:
27     #         total += num ** 2
28     #     return total
29     def optimized_sum_of_squares(numbers):
30         return sum(num ** 2 for num in numbers)
31     print(optimized_sum_of_squares([1, 2, 3])) # Expected output: 14
32     start = time.time()
33     print(optimized_sum_of_squares(numbers))
34     end = time.time()
35     print("Execution Time (Optimized):", end - start)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab10.3/task05.py
14
33333283333335000000
Execution Time (Original): 0.11085295677185059
14
33333283333335000000
Execution Time (Optimized): 0.11381196975708008
%anumandlarithika@Mac Ai_Assisted_lab %
```

Ln 16, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 Python 3.9.6