Course Title	AI Assisted Coding

Name: G. Snehith Roll Number: 2303A52324 Batch: 45

AssignmentNumber:10.3(Present assignment number)/**24**(Total number of assignments)

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 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).

Prompt: Compare the code anf fix the bug logic and give the correct logic code with brief description of what was the bug mistake

Code:

```python
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
print("Before fixing Errors: ",factorial(5))

def factorial(n):
    result = 1
    for i in range(1, n + 1):  # Changed: n + 1 to include n
        result = result * i
    return result
print("After fixing Errors: ",factorial(5))  # Output: 120
```

Output:

```
PS D:\3-2\AI Assitant Coding\Code Files>  d:; cd 'd:\3-2\AI Assitant Coding\
AI Assitant Coding\Code Files\.venv\Scripts\python.exe' 'c:\Users\SNEHITH\.v
AI Assitant Coding\Code Files\.venv\Scripts\python.exe' 'c:\Users\SNEHITH\.v
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57529' '--'
ing\Code Files\Task-10-3.py'
Before fixing Errors:  24
After fixing Errors:  120
PS D:\3-2\AI Assitant Coding\Code Files>
```

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:

1. 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.

2. Compare the original and AI-improved versions.
3. 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.

Prompt: Review the following Python function for readability and documentation:

1. Critique its readability, parameter naming, and missing documentation.

2. Rewrite it with:

- Descriptive function and parameter names

- A complete docstring (description, parameters, return, examples)

- Input validation

- Exception handling (including division by zero)

3. Compare the original and improved versions.

4. Provide test cases for valid and invalid inputs.

Code:

```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":
        return a / b
print("Before fixing Errors: ",calc(10, 3, "div"))  # Output: 2.0


# ...existing code...
def calculate(operand1: float, operand2: float, operation: str) -> float:
    """
    Perform a basic arithmetic operation on two numeric operands.

    Parameters
    ----------
    operand1 : int | float
        The first numeric operand.
```

Output:

```
PS D:\3-2\AI Assitant Coding\Code Files>  d:; cd 'd:\3-2\AI Assitant Coding\Code
AI Assitant Coding\Code Files\.venv\Scripts\python.exe' 'c:\Users\SNEHITH\.vscode
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62205' '--' 'D:\3
ing\Code Files\Task-10-3.py'
ing\Code Files\Task-10-3.py'
Before fixing Errors:  3.3333333333333335
8
2
15
2
15
15
2.0
ZeroDivisionError division by zero
TypeError operands must be int or float
ValueError Unsupported operation: 'pow'
PS D:\3-2\AI Assitant Coding\Code Files>
```

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:

1. Verify the function works correctly for sample inputs.
2. 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).
3. Apply the AI-suggested changes and verify functionality is preserved.
4. 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
```

Prompt:

Verify the function works correctly for sample inputs.

1. 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).

2. Apply the AI-suggested changes and verify functionality is preserved.

Write a short note on how automated AI reviews could streamline code reviews in large teams.

Code:

```python
def Checkprime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
print("Before Fixing:",Checkprime(12))  # True

# ...existing code...
def is_prime(n: int) -> bool:
    """
    Return True if n is a prime number, False otherwise.

    Parameters
    ----------
    n : int
        Integer to test for primality.

    Returns
    -------
    bool
        True if n is prime, False otherwise.
    """
```

Output:

```
PS D:\3-2\AI Assitant Coding\Code Files>  d:; cd 'd:\3-2\AI Assitant Coding\
AI Assitant Coding\Code Files\.venv\Scripts\python.exe' 'c:\Users\SNEHITH\.v
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54286' '--'
ing\Code Files\Task-10-3.py'
Before Fixing: False
After Fixing:
True
True
False
False
True
False
False
PS D:\3-2\AI Assitant Coding\Code Files>
```

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]
```

Prompt: Act as a professional Python code reviewer.

Review the following function:

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

1. Critique its readability, naming, reusability, and edge cases.

2. Suggest improvements including:

- Better function and variable names
- Clear purpose
- Type hints
- Input validation
- Configurable multiplier instead of hardcoded 2

3. Refactor the function based on your suggestions.

4. Write a short reflection on whether AI should act as a standalone reviewer or an assistant.

Code:

```
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
print("Before Fixing:",processData([1, 2, 3, 4, 5]))  # [4, 8]

# ...existing code...
from typing import Iterable, Union, List
Number = Union[int, float]
def scale_even_numbers(data: Iterable[Number], multiplier: Number = 2) -> List[N
    """
    Return a new list containing each even numeric element from `data` multiplie

    Parameters
    ----------
    data : Iterable[int | float]
        Iterable of numeric values.
    multiplier : int | float, optional
        Value to multiply each even element by (default 2).

    Returns
    -------
    list[int | float]
        List of scaled even numbers.
```

Output:

```
PS D:\3-2\AI Assitant Coding\Code Files>  d:; cd 'd:\3-2\AI Assitant Coding\Code F
AI Assitant Coding\Code Files\.venv\Scripts\python.exe' 'c:\Users\SNEHITH\.vscode\
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60310' '--' 'D:\3-
ing\Code Files\Task-10-3.py'
Before Fixing: [4, 8]
After Fixing: [4, 8]
PS D:\3-2\AI Assitant Coding\Code Files>
```

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)
```

Prompt: Act as a Python performance optimization expert.

Given the function:

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

1. Analyze its time complexity.

2. Suggest performance improvements (e.g., built-in functions or NumPy if appropriate).

3. Provide an optimized version.

4. Compare execution time before and after optimization using a large dataset (e.g., range(1000000)).

5. Briefly 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)

Code:

```python
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
print("Before Optimization:", sum_of_squares(range(1000000)))  # 333333166666500000


# ...existing code...
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total

def sum_of_squares_optimized(numbers):
    """Use built-in sum with generator to reduce Python-level overhead."""
    return sum(x * x for x in numbers)

def sum_of_squares_numpy(numbers):
    """NumPy vectorized version (best for large numeric arrays)."""
```

Output:

```
PS D:\3-2\AI Assitant Coding\Code Files>  d:; cd 'd:\3-2\AI Assitant Coding\Code F
cripts\python.exe' 'c:\Users\SNEHITH\.vscode\extensions\ms-python.debugpy-2025.18.
' 'D:\3-2\AI Assitant Coding\Code Files\Task-10-3.py'
Before Optimization: 333332833333500000
original: 0.10481200000504032 s
optimized (sum generator): 0.11997659999178723 s
NumPy not available
PS D:\3-2\AI Assitant Coding\Code Files>
```