

ASSIGNMENT - 10.5

NAME : S VYSHNAVI

HT NO : 2303A52239

BATCH : 35

Lab 10 – Code Review and Quality: Using AI to Improve Code Quality and Readability Week5 – Friday Lab Objectives • Use AI for automated code review and quality enhancement. • Identify and fix syntax, logical, performance, and security issues in Python code. • Improve readability and maintainability through structured refactoring and comments. • Apply prompt engineering for targeted improvements. • Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices Lab Outcomes

1. Students will be able to use AI tools to review code.
2. Students will be able to improve code quality and readability.
3. Students will be able to identify and fix common coding issues

Task Description #1 – Variable Naming Issues Task: Use AI to improve unclear variable names. Sample Input Code: def f(a, b): return a + b
print(f(10, 20)) Expected Output: • Code rewritten with meaningful function and variable names.

```
def add_numbers(first_number, second_number):
    """
    Returns the sum of two numbers.
    """
    return first_number + second_number

result = add_numbers(10, 20)
print(result)
```

30

Task Description #2 – Missing Error Handling Task: Use AI to add proper error handling. Sample Input Code: def divide(a, b): return a / b
print(divide(10, 0)) Expected Output: • Code with exception handling and clear error messages

```
def divide_numbers(numerator, denominator):
    """
    Divides two numbers and returns the result.

    Raises:
        ValueError: If the denominator is zero
    """
    try:
        return numerator / denominator
    except ZeroDivisionError:
        raise ValueError("Error: Division by zero is not allowed.")

    try:
        result = divide_numbers(10, 0)
        print(result)
    except ValueError as error:
        print(error)
```

Error: Division by zero is not allowed.

Task Description #3: Student Marks Processing System The following program calculates total, average, and grade of a student, but it has poor readability, style issues, and no error handling. marks=[78,85,90,66,88] t=0 for i in marks: t=t+i a=t/len(marks) if a>=90: print("A") elif a>=75: print("B") elif a>=60: print("C") else: print("F") Task: • Use AI to refactor the code to follow PEP 8 standards. • Add meaningful variable names, functions, and comments. • Add basic input validation and documentation.

```
from typing import List

def calculate_total(marks: List[int]) -> int:
```

```

"""
Calculate the total marks from a list of marks.

:param marks: List of student marks
:return: Total marks
"""

if not marks:
    raise ValueError("Marks list cannot be empty.")
if any(not isinstance(mark, (int, float)) or mark < 0 for mark in marks):
    raise ValueError("All marks must be non-negative numbers.")

return sum(marks)

```



```

def calculate_average(marks: List[int]) -> float:
"""
Calculate the average marks from a list of marks.

:param marks: List of student marks
:return: Average marks
"""

total = calculate_total(marks)
return total / len(marks)

```



```

def determine_grade(average: float) -> str:
"""
Determine the grade based on average marks.

:param average: Average marks
:return: Grade as a string
"""

if average >= 90:
    return "A"
elif average >= 75:
    return "B"
elif average >= 60:
    return "C"
else:
    return "F"

```



```

# Sample marks list
marks = [78, 85, 90, 66, 88]

try:
    average_marks = calculate_average(marks)
    grade = determine_grade(average_marks)
    print(f"Total Marks: {calculate_total(marks)}")
    print(f"Average Marks: {average_marks:.2f}")
    print(f"Grade: {grade}")
except ValueError as error:
    print(f"Input Error: {error}")

```

```
Total Marks: 407
Average Marks: 81.40
Grade: B
```

Task Description #4: Use AI to add docstrings and inline comments to the following function. def factorial(n): result = 1 for i in range(1,n+1): result *= i return result

```

def factorial(n: int) -> int:

    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")

    result = 1 # Initialize result to 1

    # Multiply result by each number from 1 to n
    for i in range(1, n + 1):
        result *= i # Update result with current multiplier

    return result

```

```
# Example usage
print(factorial(5)) # Output: 120

120
```

Task Description #5: Password Validation System (Enhanced) The following Python program validates a password using only a minimum length check, which is insufficient for real-world security requirements.

```
pwd = input("Enter password: ")
if len(pwd) >= 8:
    print("Strong")
else:
    print("Weak")
```

1. Enhance password validation using AI assistance to include multiple security rules such as:
 - o Minimum length requirement
 - o Presence of at least one uppercase letter
 - o Presence of at least one lowercase letter
 - o Presence of at least one digit
 - o Presence of at least one special character
2. Refactor the program to:
 - o Use meaningful variable and function names
 - o Follow PEP 8 coding standards
 - o Include inline comments and a docstring
3. Analyze the improvements by comparing the original and AI- enhanced versions in terms of:
 - o Code readability and structure
 - o Maintainability and reusability
 - o Security strength and robustness
4. Justify the AI-generated changes, explaining why each added rule and refactoring decision improves the overall quality of the program

```
import re

def is_strong_password(password: str) -> bool:
    if len(password) < 8:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[@#$%^&(*\)-_+=]", password):
        return False
    return True

# Main program
password_input = input("Enter password: ")

if is_strong_password(password_input):
    print("Strong")
else:
    print("Weak")
```

```
Enter password: Harini@4
Strong
```

Start coding or generate with AI.