AI ASSIST CODING

ASSIGNMENT-10.4

KIRANMAI VENDI

2303A52506

BATCH NO:15

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

Task 1: AI-Assisted Syntax and Code Quality Review

PROMPT:

Review the following Python script. Identify syntax errors, indentation problems, incorrect variable names, and faulty function calls. Correct them and explain each fix according to PEP 8 standards.

CODE:

```
def calculate_total(price, tax)

total = price + price * tax

return total


itemPrice = 100

TaxRate = 0.15
```

```
result = calculateTotal(itemprice,  Taxrate)

print("Total price is: ", result
```

ISSUES IDENTIFIED:

Issue Type	Problem
Syntax Error	Missing colon in function definition
Indentation	Incorrect indentation of function body
Naming Issue	calculateTotal incorrect function call
Case Sensitivity	itemprice, Taxrate inconsistent
Syntax Error	Missing closing parenthesis in print
PEP 8 Violation	Inconsistent variable naming style

CORRECTED CODE:

```
def calculate_total(price: float, tax_rate: float) -> float:
    """
    Calculate total price including tax.

    :param price: Base price of the item
    :param tax_rate: Tax rate as decimal
    :return: Total price including tax
    """
    total = price + (price * tax_rate)
    return total


item_price = 100
tax_rate = 0.15
```

```
result = calculate_total(item_price, tax_rate)

print("Total price is:", result)
```

EXPLANATION:

- Added missing colon (`:`) after function definition.
- Fixed indentation inside function.
- Added missing closing parenthesis in `print()`.
  NAMING CORRECTION:
- Standardized function name to `snake_case` (`calculate_total`).
- Changed variables to `item_price` and `tax_rate` (PEP 8 compliance).

  STRUCTURAL IMPROVMENT:
- Added type hints.
- Added docstring.
- Improved readability and formatting

Task 2 – Performance-Oriented Code Review

PROMPT:

Analyze the following function for performance bottlenecks and refactor it to improve time complexity while preserving functionality

ORIGINAL INEFFCIENT CODE:

```
def find_duplicates(numbers):
    duplicates = []
    for i in range(len(numbers)):
        for j in range(i + 1, len(numbers)):
            if numbers[i] == numbers[j]:
                if numbers[i] not in duplicates:
                    duplicates.append(numbers[i])
    return duplicates
```

PERFORMANCE PROBLEM:

- Uses **nested loops**
- Time complexity: **$O(n^2)$**
- Inefficient for large datasets

OPTIMIZED VERSION:

```
def find_duplicates(numbers):
    """
    Identify duplicate values in a list efficiently.

    :param numbers: List of numbers
    :return: List of duplicate values
    """
    seen = set()
    duplicates = set()

    for number in numbers:
        if number in seen:
            duplicates.add(number)
        else:
            seen.add(number)

    return list(duplicates)
```

PERFORMANCE EXPLANATION:

Version	Time Complexity	Reason
Original	$O(n^2)$	Nested loops compare each element
Optimized	O(n)	Uses hash-based lookup (set)

Task 3: Readability and Maintainability Refactoring

PROMPT:

Refactor this function for clarity, apply PEP 8 formatting, improve naming conventions, and add documentation

ORIGINAL CODE:

```
def f(x,y):
z=[]
for i in x:
  if i>y:
   z.append(i)
return z
```

REFACTORED VERSION:

```
def filter_greater_than(values: list, threshold: int) -> list:
    filtered_values = []

    for value in values:
        if value > threshold:
            filtered_values.append(value)
```

```
    return filtered_values
```

READABILITY IMPROVEMENTS:
IMPROVMENTS:

- Renamed function `f` → `filter_greater_than`
- Renamed variables `x`, `y`, `z` → meaningful names
- Fixed indentation
- Added type hints
- Added docstring
- Added spacing for readability

MAINTAINABILITY:

- Easier for future developers to understand
- Self-documenting function
- Clear input/output structure

Task 4: Secure Coding and Reliability Review

Original Insecure Code:

```
import sqlite3

def get_user(username):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE username = '" + username + "'"
    cursor.execute(query)
    result = cursor.fetchone()
    conn.close()
    return result
```

Security Issues Identified:

Issue	Risk

SQL string concatenation	SQL Injection
No input validation	Malicious input
No exception handling	Application crash
No connection safety	Resource leaks

SECURE VERSION:

```
import sqlite3

def get_user(username: str):
    if not isinstance(username, str) or not username.strip():
        raise ValueError("Invalid username provided.")


    try:
        with sqlite3.connect("users.db") as connection:
            cursor = connection.cursor()
            query = "SELECT * FROM users WHERE username = ?"
            cursor.execute(query, (username,))
            return cursor.fetchone()


    except sqlite3.Error as error:
        print("Database error occurred:", error)
        return None
```

SECURITY IMPROVEMENTS:

Parameterized Queries

- Prevents SQL injection
- Uses ? placeholder

Input Validation

Ensures username is non-empty string

Exception Handling

- Prevents runtime crashes
- Handles database errors safely

Context Manager (`with`)

Ensures connection closes automatically

Task 5: AI-Based Automated Code Review Report:

Sample Script Reviewed

def calc(a,b):return a+b

x=10

y=20

print(calc(x,y))

AI Code Review:

Readability:

Single-line function reduces readability
No whitespace formatting

Recomandation:

- Expand function body
- Apply proper indentation

Naming Conventions:

`calc`, `a`, `b`, `x`, `y` are not descriptive

Recomandation:

- Use meaningful names like `add_numbers`, `num1`, `num2`

Formatting & Style (PEP 8):

No spacing around operators
No blank lines between sections

Error Handling

No input validation
No type checking

Documentation Quality:

No docstring
No comments

Maintainability:

Risk Level: **Medium**

Reasons:

- Poor naming
- No documentation
- Hard to extend

IMPROVED VERSION:

```
def add_numbers(num1: int, num2: int) -> int:
    """
    Return the sum of two integers.

    :param num1: First integer
    :param num2: Second integer
    :return: Sum of num1 and num2
    """
```

```
    return num1 + num2



first_number = 10

second_number = 20


print(add_numbers(first_number, second_number))
```