

Assignment 10.4  
2303A510B7 - Harini.S  
Batch - 14

## Lab 9 – Code Review and Quality: Using AI to Improve Code Quality and Readability

### Task 1: AI-Assisted Syntax and Code Quality Review

Wrong Code:

```
def calculate_total(price,quantity)  
total = price * quantity  
Print("Total is:", total)  
  
calculateTotal(10,5)
```

Output:

```
File "c:\Users\danda\OneDrive\Documents\3.2\AIA\AIAC.py", line 1  
def calculate_total(price,quantity)  
                                ^  
SyntaxError: expected ':'
```

Prompt:

Review the following Python code. Identify syntax errors, indentation issues, incorrect variable names, and faulty function calls. Correct the code and explain each fix clearly. Ensure final code follows PEP 8 standards.

## Corrected Code:

```
def calculate_total(price, quantity):  
    """  
    Calculate the total cost based on price and quantity.  
    """  
    total = price * quantity  
    print("Total is:", total)  
    return total  
calculate_total(10, 5)  
calculate_total(20, 3)  
calculate_total(15, 7)
```

## Output:

```
Total is: 50  
Total is: 60  
Total is: 105
```

## Explanation:

- The corrected Python code calculates the total cost by multiplying price and quantity, prints the result, and returns the value.
- It follows PEP 8 standards with proper indentation, spacing, and naming.
- The function is defined with parameters, indented body, and a return statement.
- The call passes arguments 10 and 5, outputting "Total is: 50".
- This ensures the code is syntactically correct and reusable

## Task 2: Performance-Oriented Code Review

### Prompt:

Analyze this function for performance bottlenecks. Optimize duplicate detection logic. Explain time complexity improvement.

### Code:

```
def find_duplicates(data):
    duplicates = []
    for i in range(len(data)):
        for j in range(i + 1, len(data)):
            if data[i] == data[j] and data[i] not in duplicates:
                duplicates.append(data[i])
    return duplicates
data = [1, 2, 3, 4, 5, 2, 3, 6]
duplicates = find_duplicates(data)
print("Duplicates:", duplicates)
```

### Output:

```
Duplicates: [2, 3]
```

### Explanation:

- **Time Complexity:** Reduced from  $O(n^2)$  to  $O(n)$  by using a single pass through the data with set operations (average  $O(1)$  per operation). This scales much better for large inputs.
- **Space Complexity:** Now  $O(n)$  in the worst case (for seen and duplicates sets), but acceptable for the performance gain.
- **Logic Changes:**
  - seen tracks elements already encountered.
  - duplicates collects items that appear more than once.
  - Eliminates nested loops and list scans.

- **Trade-offs:** Sets don't preserve order; if order matters, you could use a list for duplicates and check if item not in duplicates (still  $O(n)$  overall but with higher constants). For hashable data only (e.g., not lists/dicts as elements). If data contains unhashable types, fall back to a dict-based approach.

## Task 3: Readability and Maintainability Refactoring

### Prompt:

Refactor the following code to improve readability. Apply PEP 8 standards. Improve naming and add documentation.

### Wrong Code:

```
def f(a,b):  
    x=a+b  
    y=a-b  
    z=a*b  
    return x,y,z  
a=10  
b=5  
x,y,z=f(a,b)  
print("Sum:",x)  
print("Difference:",y)  
print("Product:",z)
```

### Output:

```
Sum: 15  
Difference: 5  
Product: 50
```

## Corrected Code:

```
def calculate_operations(first_number, second_number):  
    """  
    Calculate the sum, difference, and product of two numbers.  
    Args:  
        first_number (float or int): The first number.  
        second_number (float or int): The second number.  
    Returns:  
        tuple: A tuple containing (sum, difference, product).  
    """  
    sum_result = first_number + second_number  
    difference = first_number - second_number  
    product = first_number * second_number  
    return sum_result, difference, product  
first_num = 10  
second_num = 5  
sum_val, diff_val, prod_val = calculate_operations(first_num, second_num)  
print("Sum:", sum_val)  
print("Difference:", diff_val)  
print("Product:", prod_val)
```

## Output:

```
Sum: 15  
Difference: 5  
Product: 50
```

## Explanation:

1. **Function Naming:** Renamed `f` to `calculate_operations` for clarity on its purpose.
2. **Parameter Naming:** Changed `a` and `b` to `first_number` and `second_number` to be descriptive.
3. **Variable Naming:**
  - a. Inside the function: `x, y, z` → `sum_result, difference, product`.
  - b. Outside: `a, b` → `first_num, second_num`; `x, y, z` → `sum_val, diff_val, prod_val`.
4. **Documentation:** Added a docstring explaining the function's purpose, arguments, and return value.

## 5. PEP 8 Compliance:

- a. **Indentation:** Changed from 3 spaces to 4 spaces.
  - b. **Spacing:** Added spaces around operators (+, -, \*, =) and after commas in function calls/returns.
  - c. **Line length:** Kept lines concise.
6. **Readability Enhancements:** Used tuple unpacking clearly and improved print statements for better flow.

## Task 4: Secure Coding and Reliability Review

### Prompt:

Identify SQL injection vulnerabilities. Refactor code using parameterized queries. Add input validation and exception handling.

### Wrong 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.fetchall()
    conn.close()
    return result
```

## Corrected Code:

```
import sqlite3

def get_user(username):
    """
    Retrieve user data from the database by username.
    Args:
        username (str): The username to search for.
    Returns:
        list: A list of tuples containing user data, or empty list if not found.
    Raises:
        ValueError: If username is invalid.
        sqlite3.Error: If a database error occurs.
    """
    if not isinstance(username, str) or not username.strip():
        raise ValueError("Username must be a non-empty string.")
    try:
        with sqlite3.connect("users.db") as conn:
            cursor = conn.cursor()
            query = "SELECT * FROM users WHERE username = ?"
            cursor.execute(query, (username,))
            result = cursor.fetchall()
            return result
    except sqlite3.Error as e:
        print(f"Database error: {e}")
        raise
```

## Explanation:

- **Parameterized Queries:** Replaced string concatenation with ? placeholder and passed username as a tuple to cursor.execute(). This automatically escapes input and prevents SQL injection.
- **Input Validation:** Added checks to ensure username is a non-empty string.
- **Exception Handling:** Wrapped the code in try-except to catch sqlite3.Error and re-raise after logging. Used with statement for the connection to ensure it's closed even on errors.
- **Documentation:** Added a docstring explaining the function, args, returns, and raises.
- **PEP 8 Compliance:** Proper indentation, spacing, and naming.

## Task 5: AI-Based Automated Code Review Report

### Prompt:

Generate a structured code review report for the following Python script. Evaluate readability, naming, formatting, documentation, maintainability, and risk areas. Provide improvement suggestions.

### Wrong Code:

```
def d(x,y):return x+y
print(d(1,2))
```

### Output:

```
3
```

### Corrected Code:

```
def add_two_numbers(first_number, second_number):
    """
    Calculate the sum of two numbers.
    Args:
        first_number (int or float): The first number to add.
        second_number (int or float): The second number to add.
    Returns:
        int or float: The sum of the two numbers.
    Raises:
        TypeError: If inputs are not numeric.
    """
    if not isinstance(first_number, (int, float)) or not isinstance(second_number, (int, float)):
        raise TypeError("Both inputs must be numbers.")
    return first_number + second_number

# Example usage:
if __name__ == "__main__":
    num1 = 5
    num2 = 10
    result = add_two_numbers(num1, num2)
    print(f"The sum of {num1} and {num2} is: {result}")
```

### Output:

```
The sum of 5 and 10 is: 15
```



## Explanation:

- The original code `def d(x,y):return x+y` defines a function to add two numbers.
- It uses cryptic naming: 'd' for the function and 'x', 'y' for parameters, lacking clarity.
- Formatting violates PEP 8 with no spaces around operators or after commas.
- There's no documentation, making it hard for others to understand or maintain.
- Readability suffers from everything on one line, reducing scannability.
- Maintainability is low due to no error handling or type checks.
- Risks include runtime errors if non-numeric inputs are passed.
- The refactored version improves naming, adds docstrings, and includes validation.
- It follows PEP 8 with proper spacing and structure.

