AI Assistant Coding
Assignment-10.4

Name:G.Harika
Hall Ticket No.:2303A51612
Batch No.:05

Task 1: AI-Assisted Syntax and Code Quality Review

Scenario

You join a development team and are asked to review a junior developer’s Python script that fails to run correctly due to basic coding mistakes. Before deployment, the code must be corrected and standardized.

Task Description

You are given a Python script containing:

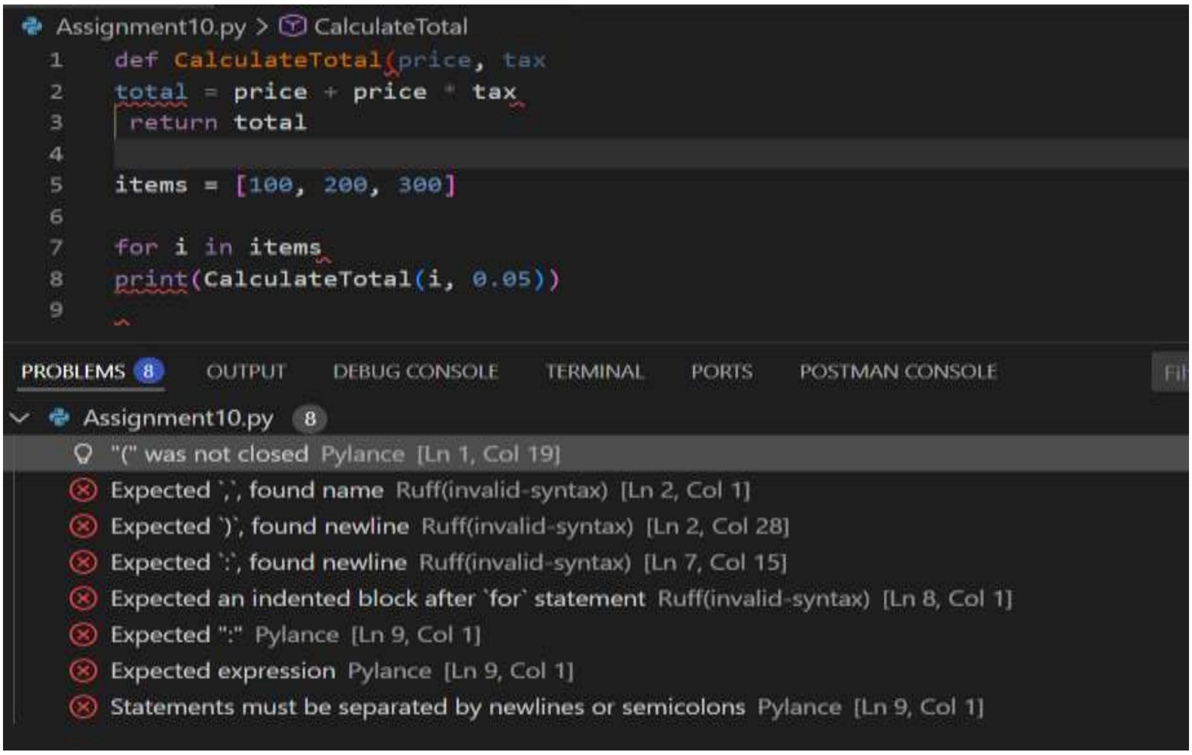
- Syntax errors
- Indentation issues
- Incorrect variable names
- Faulty function calls

Use an AI tool (GitHub Copilot / Cursor AI) to:

- Identify all syntactic and structural errors
- Correct them systematically
- Generate an explanation of each fix made

Python Script with errors:

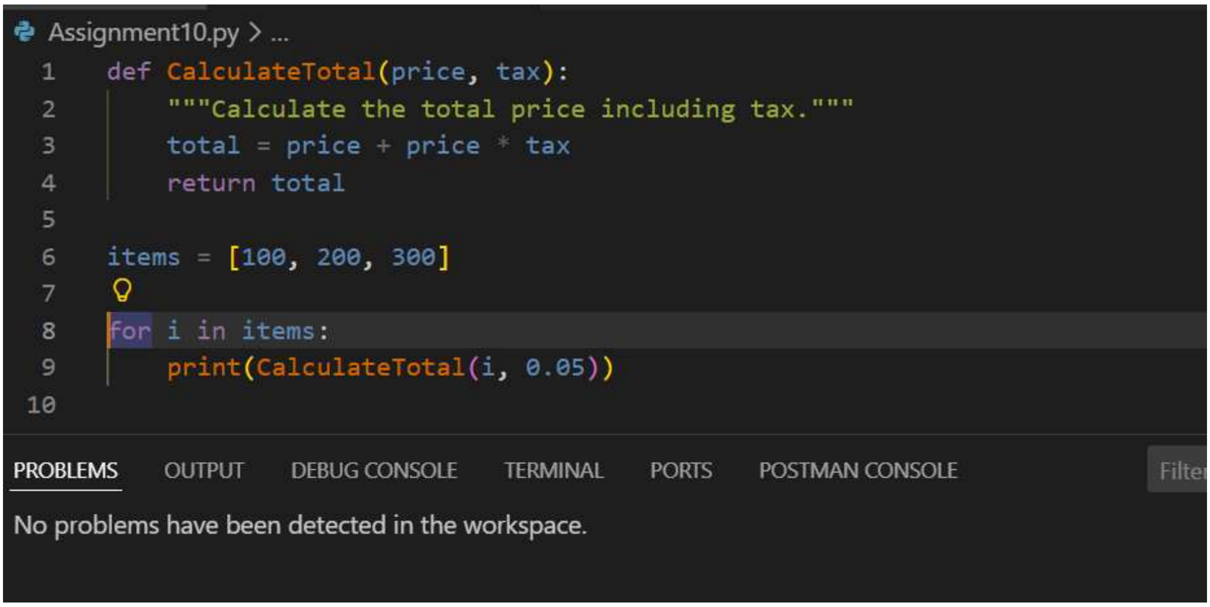
```
def CalculateTotal(price, tax
total = price + price * tax
 return total

items = [100, 200, 300]

for i in items
print(CalculateTotal(i, 0.05))
```



```
def CalculateTotal(price, tax
total = price + price * tax
 return total

items = [100, 200, 300]

for i in items
print(CalculateTotal(i, 0.05))
```

```
"(" was not closed Pylance [Ln 1, Col 19]
Expected `,`, found name Ruff(invalid-syntax) [Ln 2, Col 1]
Expected `)`, found newline Ruff(invalid-syntax) [Ln 2, Col 28]
Expected `:`, found newline Ruff(invalid-syntax) [Ln 7, Col 15]
Expected an indented block after `for` statement Ruff(invalid-syntax) [Ln 8, Col 1]
Expected ":" Pylance [Ln 9, Col 1]
Expected expression Pylance [Ln 9, Col 1]
Statements must be separated by newlines or semicolons Pylance [Ln 9, Col 1]
```

Output:



```
def CalculateTotal(price, tax):
    """Calculate the total price including tax."""
    total = price + price * tax
    return total

items = [100, 200, 300]

for i in items:
    print(CalculateTotal(i, 0.05))
```

No problems have been detected in the workspace.

AI Explanation of Fixes

Syntax Fixes

- Added missing) in function definition.
- Corrected indentation inside the function.
- Added : after the for loop.
- Fixed indentation for the print() statement.

Naming Corrections

- Renamed CalculateTotal → calculate_total

  ✓ Follows **PEP 8 snake_case convention**.
- Renamed loop variable i → item for clarity.

Structural Improvements

- Added a **docstring** explaining the function.
- Improved spacing for readability.

Task 2: Performance-Oriented Code Review

Scenario

A data processing function works correctly but is inefficient and slows down the system when large datasets are used.

Task Description

You are provided with a function that identifies duplicate values in a list using inefficient nested loops.

Using AI-assisted code review:

• Analyze the logic for performance bottlenecks

• Refactor the code for better time complexity

• Preserve the correctness of the output

Ask the AI to explain:

• Why the original approach was inefficient

• How the optimized version improves performance

Inefficient Duplicate Finder

```
def find_duplicates(lst):
    duplicates = []
    for i in range(len(lst)):
        for j in range(i + 1, len(lst)):
            if lst[i] == lst[j]:
                duplicates.append(lst[i])
    return duplicates
```

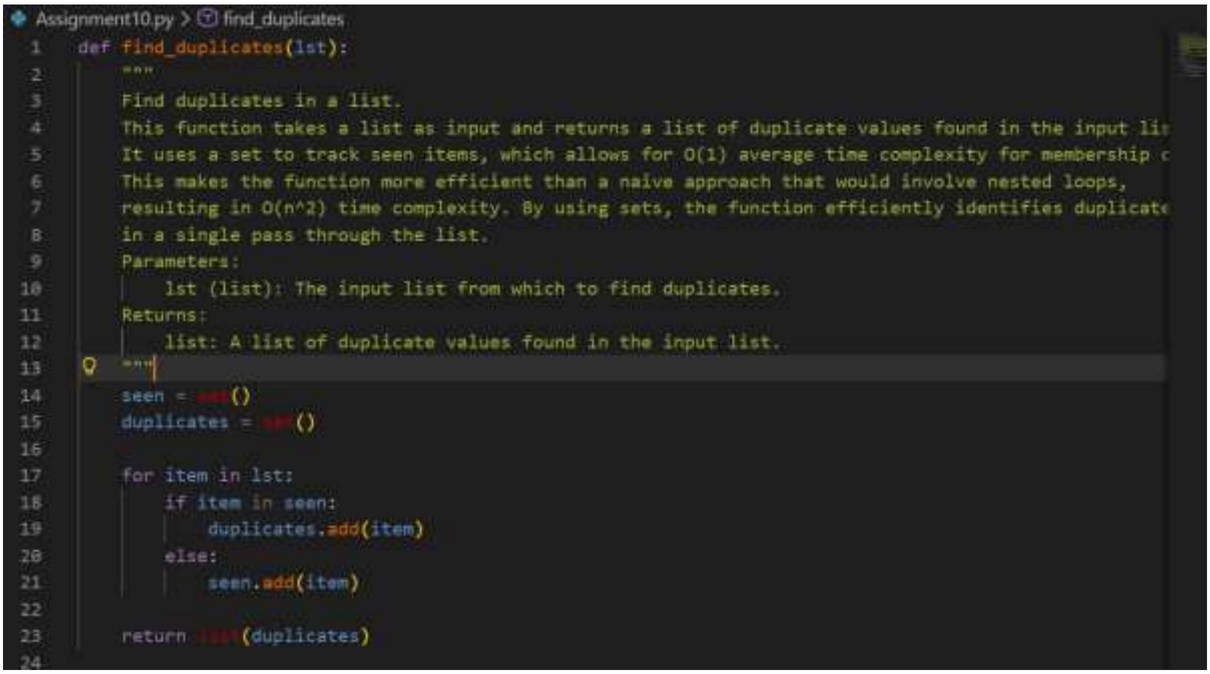
Optimised Code:

```
def find_duplicates(lst):
    """
    Find duplicates in a list.
    This function takes a list as input and returns a list of duplicate values found in the input list.
    It uses a set to track seen items, which allows for O(1) average time complexity for membership checks.
    This makes the function more efficient than a naive approach that would involve nested loops,
    resulting in O(n^2) time complexity. By using sets, the function efficiently identifies duplicates
    in a single pass through the list.
    Parameters:
        lst (list): The input list from which to find duplicates.
    Returns:
        list: A list of duplicate values found in the input list.
    """
    seen = set()
    duplicates = set()

    for item in lst:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)

    return list(duplicates)
```

Problem Analysis

Why It Is Inefficient

- Uses **nested loops**.
- Time Complexity = **O(n²)**.
- Extremely slow for large datasets.



Assignment10.py > find_duplicates

```python
def find_duplicates(lst):
    """
    Find duplicates in a list.
    This function takes a list as input and returns a list of duplicate values found in the input lis
    It uses a set to track seen items, which allows for O(1) average time complexity for membership c
    This makes the function more efficient than a naive approach that would involve nested loops,
    resulting in O(n^2) time complexity. By using sets, the function efficiently identifies duplicate
    in a single pass through the list.
    Parameters:
        lst (list): The input list from which to find duplicates.
    Returns:
        list: A list of duplicate values found in the input list.
    """
    seen = set()
    duplicates = set()

    for item in lst:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)

    return list(duplicates)
```

Performance Improvement Explanation

Approach	Time Complexity
Nested loops	$O(n^2)$
Hash set	**O(n)**

Why Faster?

- Set lookups are **O(1)**.
- Only one loop is required.

Task 3: Readability and Maintainability Refactoring

Scenario

A working script exists in a project, but it is difficult to understand due to poor naming, formatting, and structure. The team wants it rewritten for long-term maintainability.

Task Description

You are given a poorly structured Python function with:

- Cryptic function names
- Poor indentation
- Unclear variable naming
- No documentation

Use AI-assisted review to:

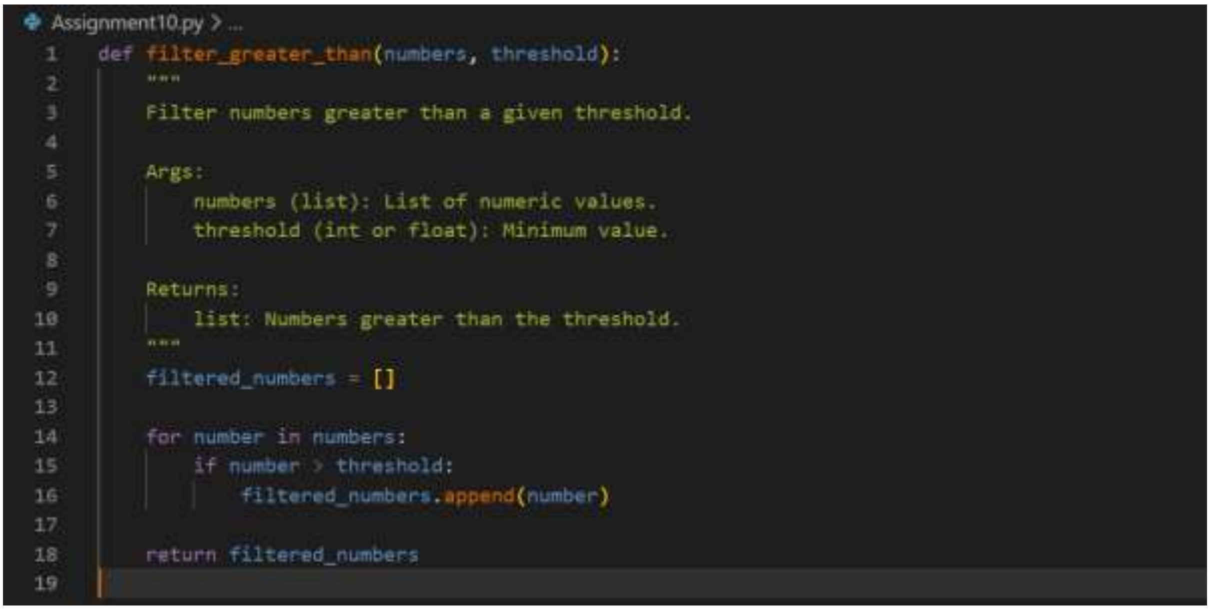
- Refactor the code for clarity
- Apply PEP 8 formatting standards
- Improve naming conventions
- Add meaningful documentation

Poorly Written Function

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

Refactored Version

```
def filter_greater_than(numbers, threshold):
    """
    Filter numbers greater than a given threshold.

    Args:
        numbers (list): List of numeric values.
        threshold (int or float): Minimum value.

    Returns:
        list: Numbers greater than the threshold.
    """
    filtered_numbers = []

    for number in numbers:
        if number > threshold:
            filtered_numbers.append(number)

    return filtered_numbers
```



```python
def filter_greater_than(numbers, threshold):
    """
    Filter numbers greater than a given threshold.

    Args:
        numbers (list): List of numeric values.
        threshold (int or float): Minimum value.

    Returns:
        list: Numbers greater than the threshold.
    """
    filtered_numbers = []

    for number in numbers:
        if number > threshold:
            filtered_numbers.append(number)

    return filtered_numbers

```

Readability Improvements

Naming

- f → filter_greater_than
- x → numbers
- y → threshold
- z → filtered_numbers

Format

- Proper indentation.
- Logical spacing.

Documentation

- Added professional docstring.

Maintainable, team-friendly code.

Task 4: Secure Coding and Reliability Review

Scenario

A backend function retrieves user data from a database but has security vulnerabilities and poor error handling, making it unsafe for production deployment.

Task Description

You are given a Python script that:

• Uses unsafe SQL query construction
• Has no input validation
• Lacks exception handling

Use AI tools to:

- Identify security vulnerabilities
- Refactor the code using safe coding practices
- Add proper exception handling
- Improve robustness and reliability

Unsafe Script

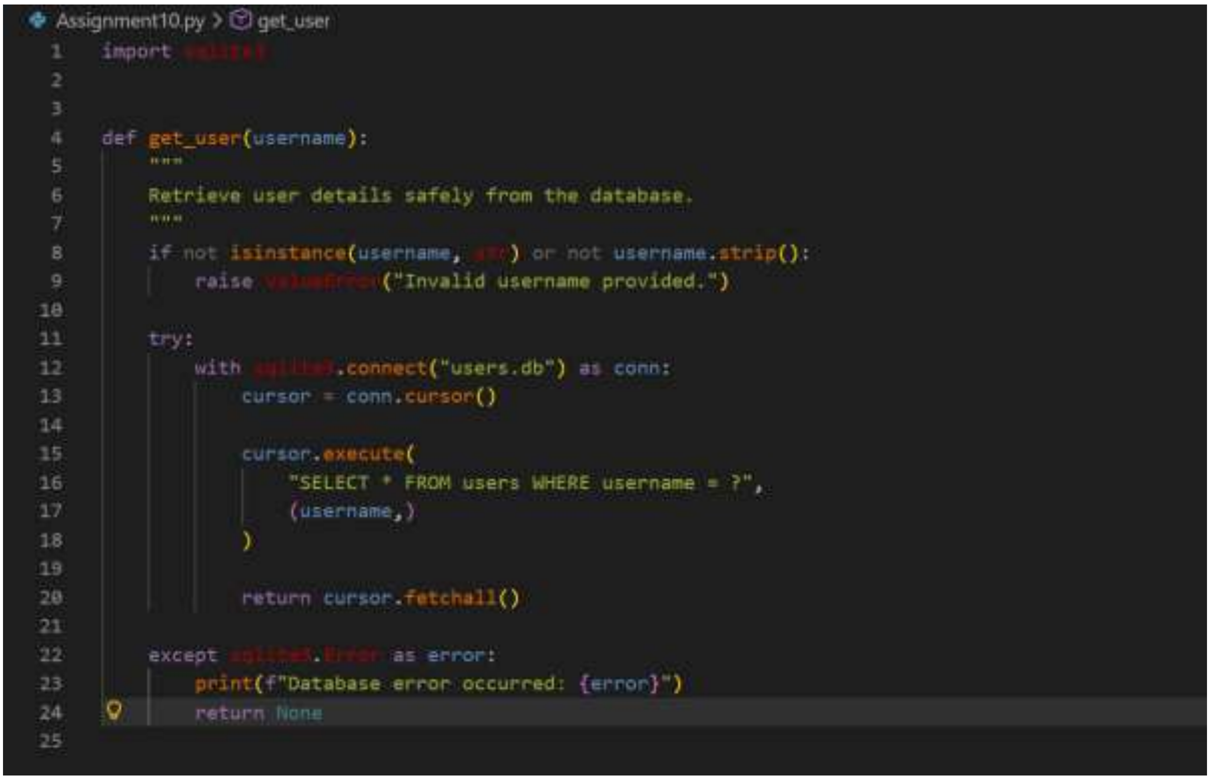
```
import sqlite3
def get_user(username):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE username = '" + username + "'"
    cursor.execute(query)
    return cursor.fetchall()
```

Security Issues Identified

- SQL Injection vulnerability.
- No input validation.
- No exception handling.
- Connection not closed.

Production-Ready Secure Version

```
import sqlite3

def get_user(username):
    """
    Retrieve user details safely from the database.
    """
    if not isinstance(username, str) or not username.strip():
        raise ValueError("Invalid username provided.")

    try:
        with sqlite3.connect("users.db") as conn:
            cursor = conn.cursor()

            cursor.execute(
                "SELECT * FROM users WHERE username = ?",
                (username,)
            )

            return cursor.fetchall()
    except sqlite3.Error as error:
        print(f"Database error occurred: {error}")
        return None
```



```python
import sqlite3


def get_user(username):
    """
    Retrieve user details safely from the database.
    """
    if not isinstance(username, str) or not username.strip():
        raise ValueError("Invalid username provided.")

    try:
        with sqlite3.connect("users.db") as conn:
            cursor = conn.cursor()

            cursor.execute(
                "SELECT * FROM users WHERE username = ?",
                (username,)
            )

            return cursor.fetchall()

    except sqlite3.Error as error:
        print(f"Database error occurred: {error}")
        return None

```

Security Improvements Explained

Parameterized Queries

Prevents SQL injection.

Input Validation

Stops malicious or empty input.

Exception Handling

Avoids runtime crashes.

Context Manager

Auto-closes database connection.

Task 5: AI-Based Automated Code Review Report

Scenario

Your team uses AI tools to perform automated preliminary code reviews before human review, to improve code quality and consistency across projects.

Task Description

You are provided with a poorly written Python script.

Using AI-assisted review:

- Generate a structured code review report that evaluates:
  - Code readability
  - Naming conventions
  - Formatting and style consistency
  - Error handling
  - Documentation quality
  - Maintainability

The task is not just to fix the code, but to analyze and report on quality issues.

Poor Script for Review

```
def d(a,b):return a/b

x=10
y=0
print(d(x,y))
```

AI Code Review Report

Readability

Issue: Cryptic function name d.

Risk: Hard for teams to understand.

Naming Conventions

Not following PEP 8.

Examples:

- a, b
- d

Use descriptive names.

Formatting & Style

- One-line function reduces readability.
- No spacing.

Error Handling

- **Critical Risk**
- Division by zero causes crash.

Documentation Quality

- None present.

Maintainability

- Low — unclear intent and unsafe execution.

```
def divide_numbers(dividend, divisor):
    '''Parameters:
        dividend (float): The number to be divided.
        divisor (float): The number by which to divide.
    Returns:
        float: The result of the division.
        ValueError: If divisor is zero, indicating that division by zero is not allowed.'''
    if divisor == 0:
        raise ValueError("Division by zero is not allowed.")
    return dividend / divisor
# Example usage:
try:
    result = divide_numbers(10, 0)
    print(result)
except ValueError as error:
    print(error)

```

Code Smells Detected

- Magic values
- Poor naming
- Missing error handling
- Lack of documentation

Demonstration of AI as Code Reviewer

This review showcased how AI can:

Detect syntax errors, Improve performance, Refactor for readability, Harden security
& Generate structured review reports.