AIAC-10.4

Name: Siri Vempati

Htno:2303A51309

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

Expected Outcome

• Fully corrected and executable Python code

• AI-generated explanation describing:

o Syntax fixes

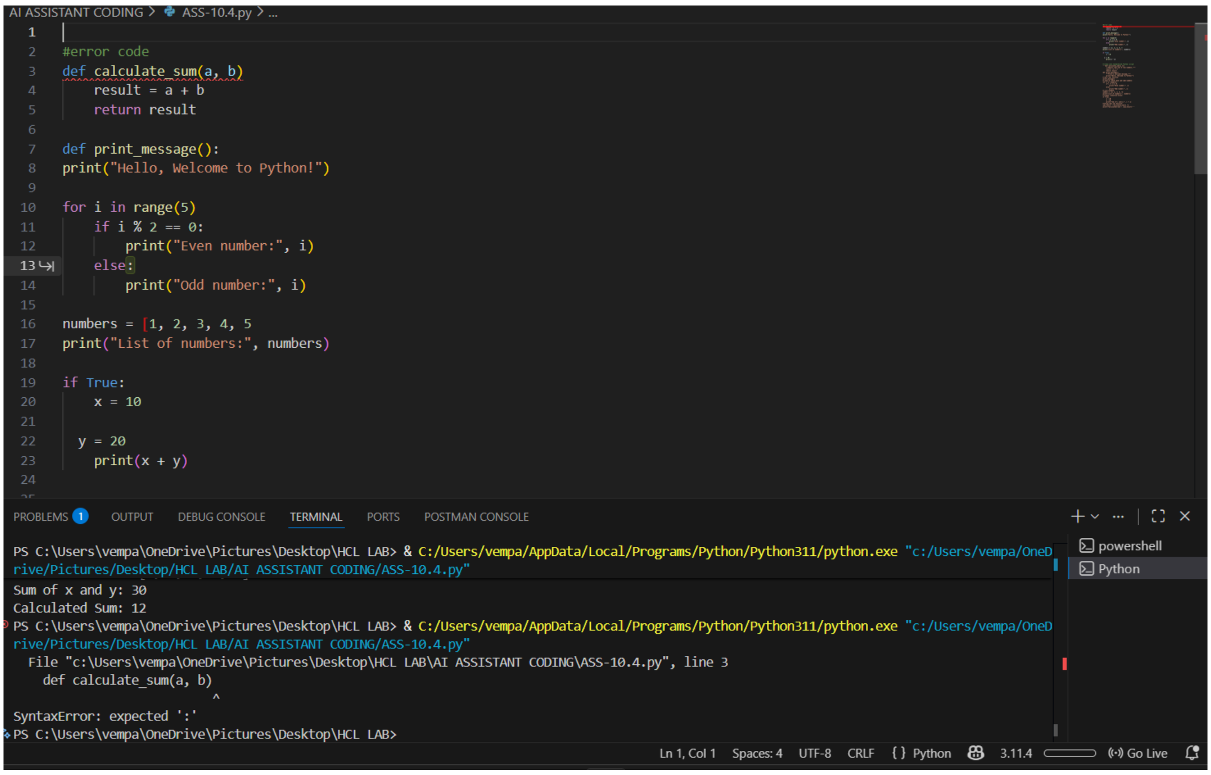
o Naming corrections

o Structural improvements

• Clean, readable version of the script

Error Code:

```
def calculate_sum(a, b)
    result = a + b
    return result
def print_message():
print("Hello, Welcome to Python!")
for i in range(5)
    if i % 2 == 0:
        print("Even number:", i)
    else
        print("Odd number:", i)
numbers = [1, 2, 3, 4, 5
print("List of numbers:", numbers)
if True:
    x = 10
  y = 20
    print(x + y)
```



```

#error code
def calculate_sum(a, b)
    result = a + b
    return result

def print_message():
print("Hello, Welcome to Python!")

for i in range(5)
    if i % 2 == 0:
        print("Even number:", i)
    else:
        print("Odd number:", i)

numbers = [1, 2, 3, 4, 5
print("List of numbers:", numbers)

if True:
    x = 10

  y = 20
    print(x + y)

```

```
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
Sum of x and y: 30
Calculated Sum: 12
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
  File "c:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB\AI ASSISTANT CODING\ASS-10.4.py", line 3
    def calculate_sum(a, b)
                           ^
SyntaxError: expected ':'
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

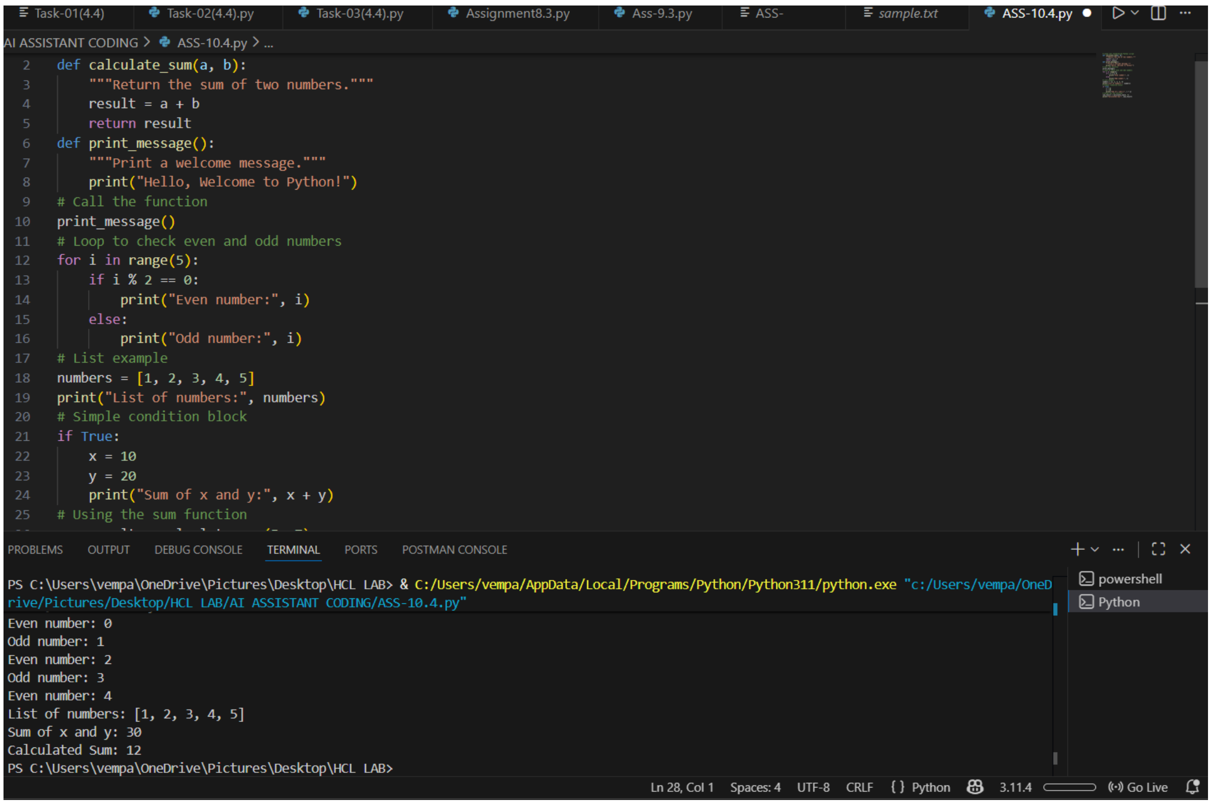
Corrected Code:

```
# Clean Python program after fixing syntax and structural errors

def calculate_sum(a, b):

    result = a + b

    return result

def print_message():

    print("Hello, Welcome to Python!")   # fixed indentation

print_message()

for i in range(5):   # added missing colon

    if i % 2 == 0:

        print("Even number:", i)

    else:   # added missing colon

        print("Odd number:", i)

numbers = [1, 2, 3, 4, 5]   # closed the list bracket

print("List of numbers:", numbers)

if True:

    x = 10

    y = 20   # fixed inconsistent indentation

    print(x + y)

# Calling the function

sum_result = calculate_sum(5, 7)

print("Sum:", sum_result)
```

Explanation(Fixes &Code):

- calculate_sum(a, b) → A function that adds two numbers and returns the result.
- print_message() → Displays a welcome message when called.
- For loop (range(5)) → Iterates from 0 to 4 and checks whether each number is even or odd using i % 2.
- List (numbers) → Stores values from 1 to 5 and prints them.

- Conditional block (if True) → Always runs, adds x and y, and prints their sum.
- Function call → calculate_sum(5, 7) returns 12, which is stored in sum_result and printed.



```python
def calculate_sum(a, b):
    """Return the sum of two numbers."""
    result = a + b
    return result
def print_message():
    """Print a welcome message."""
    print("Hello, Welcome to Python!")
# Call the function
print_message()
# Loop to check even and odd numbers
for i in range(5):
    if i % 2 == 0:
        print("Even number:", i)
    else:
        print("Odd number:", i)
# List example
numbers = [1, 2, 3, 4, 5]
print("List of numbers:", numbers)
# Simple condition block
if True:
    x = 10
    y = 20
    print("Sum of x and y:", x + y)
# Using the sum function
```

```
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
Even number: 0
Odd number: 1
Even number: 2
Odd number: 3
Even number: 4
List of numbers: [1, 2, 3, 4, 5]
Sum of x and y: 30
Calculated Sum: 12
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

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

Expected Outcome

• Optimized duplicate-detection logic (e.g., using sets or hash-

based structures)

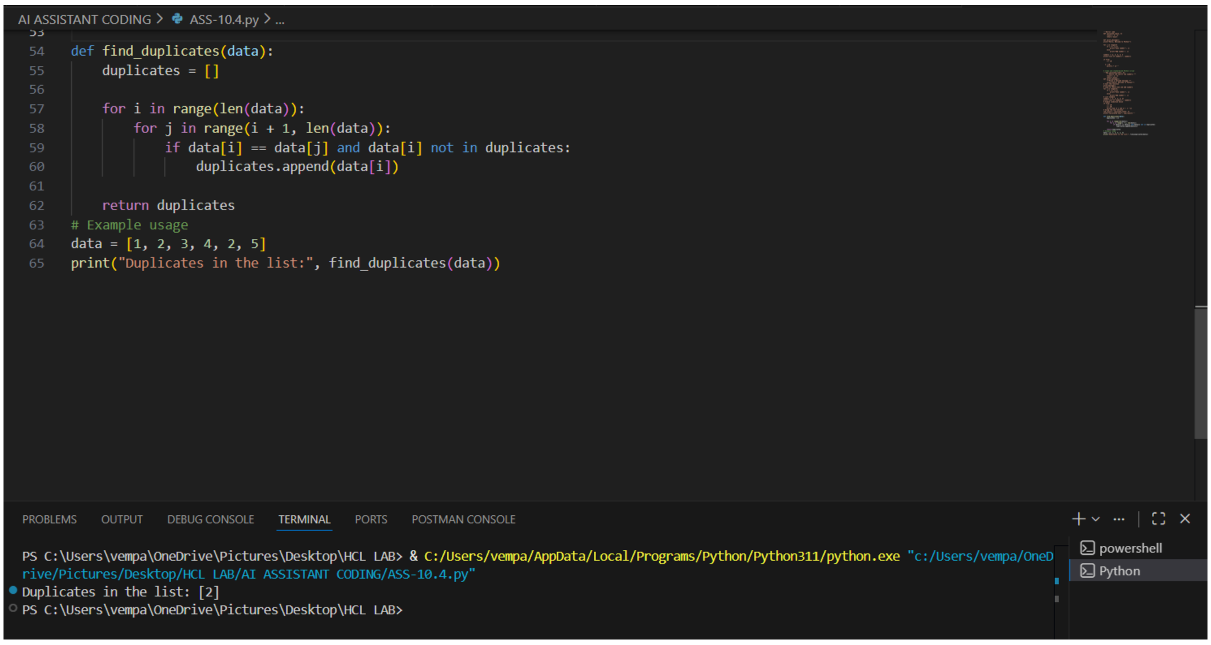
• Improved time complexity

• AI explanation of performance improvement

• Clean, readable implementation

Original 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
# Example usage
data = [1, 2, 3, 4, 2, 5]
print("Duplicates in the list:", find_duplicates(data))
```

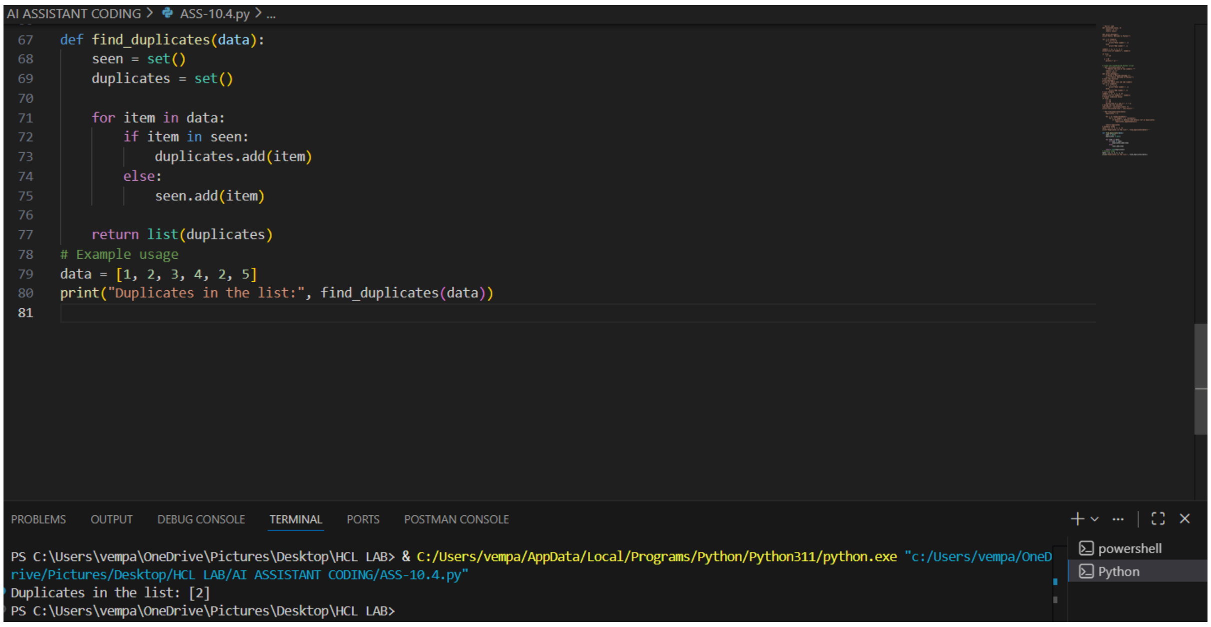


```
def find_duplicates(data):
    duplicates = []

    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
# Example usage
data = [1, 2, 3, 4, 2, 5]
print("Duplicates in the list:", find_duplicates(data))
```

```
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
Duplicates in the list: [2]
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

Optimised Code:

```
def find_duplicates(data):
    seen = set()
    duplicates = set()
    for item in data:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)
    return list(duplicates)
# Example usage
data = [1, 2, 3, 4, 2, 5]
print("Duplicates in the list:", find_duplicates(data))
```



```
def find_duplicates(data):
    seen = set()
    duplicates = set()

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

    return list(duplicates)
# Example usage
data = [1, 2, 3, 4, 2, 5]
print("Duplicates in the list:", find_duplicates(data))
```

```
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
Duplicates in the list: [2]
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

Explanation:

Optimized Version is Better

- Uses sets (hash-based structure) for fast lookup.
- Eliminates nested loops.
- Reduces time complexity from $O(n^2) \rightarrow O(n)$.

- Produces the same correct output but much faster.
- Clean, readable, and scalable implementation.

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

Expected Outcome

• Clean, well-structured code

• Descriptive function and variable names

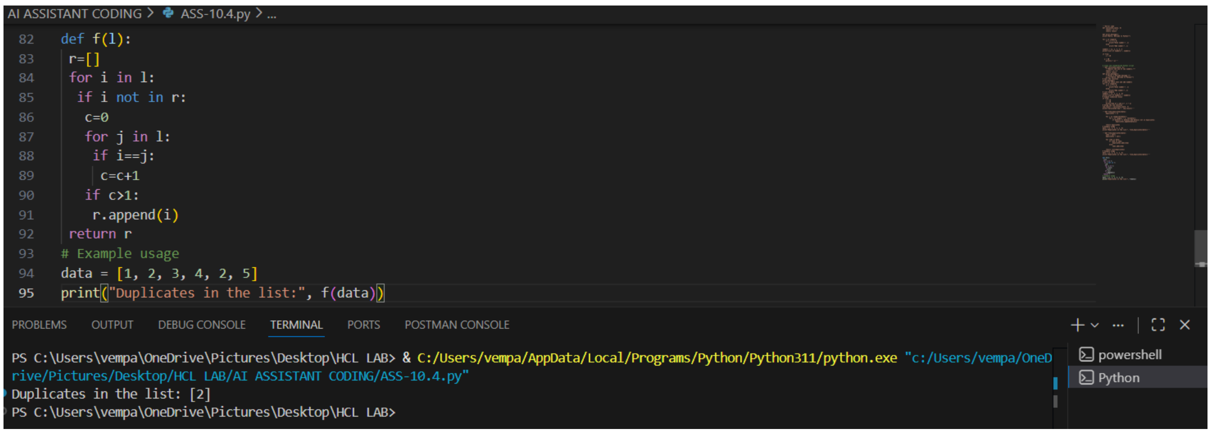
• Proper indentation and formatting

• Docstrings explaining the function purpose

• AI explanation of readability improvements

<u>Code:</u>

```
def f(l):
 r=[]
 for i in l:
  if i not in r:
   c=0
   for j in l:
    if i==j:
     c=c+1
   if c>1:
    r.append(i)
 return r
# Example usage
data = [1, 2, 3, 4, 2, 5]
print("Duplicates in the list:", f(data))
```

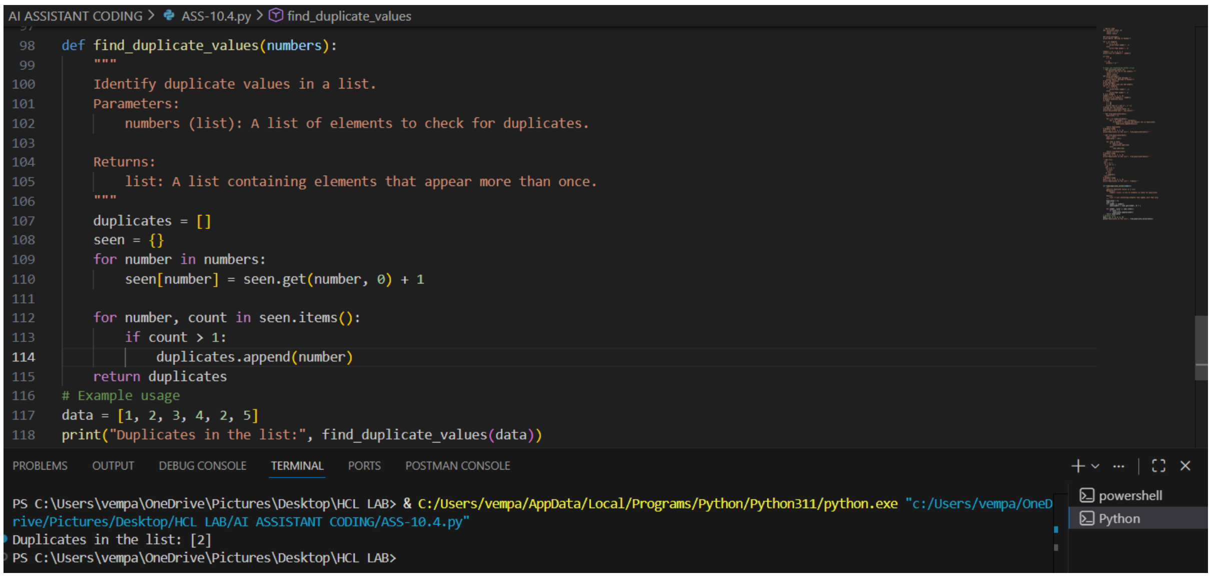

AI ASSISTANT CODING > ASS-10.4.py > ...

```
def f(l):
 r=[]
 for i in l:
  if i not in r:
   c=0
   for j in l:
    if i==j:
     c=c+1
   if c>1:
    r.append(i)
 return r
# Example usage
data = [1, 2, 3, 4, 2, 5]
print("Duplicates in the list:", f(data))
```

```
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
Duplicates in the list: [2]
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

Refactored Code:

```
def find_duplicate_values(numbers):
    """
    Identify duplicate values in a list.
    Parameters:
        numbers (list): A list of elements to check for duplicates.

    Returns:
        list: A list containing elements that appear more than once.
    """
    duplicates = []
    seen = {}
    for number in numbers:
        seen[number] = seen.get(number, 0) + 1
    for number, count in seen.items():
        if count > 1:
            duplicates.append(number)
    return duplicates
# Example usage
data = [1, 2, 3, 4, 2, 5]
print("Duplicates in the list:", find_duplicate_values(data))
```



```
def find_duplicate_values(numbers):
    """
    Identify duplicate values in a list.
    Parameters:
        numbers (list): A list of elements to check for duplicates.

    Returns:
        list: A list containing elements that appear more than once.
    """
    duplicates = []
    seen = {}
    for number in numbers:
        seen[number] = seen.get(number, 0) + 1

    for number, count in seen.items():
        if count > 1:
            duplicates.append(number)
    return duplicates
# Example usage
data = [1, 2, 3, 4, 2, 5]
print("Duplicates in the list:", find_duplicate_values(data))
```

```
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
Duplicates in the list: [2]
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

Explanation:

The refactored function find_duplicates(numbers) identifies elements that appear more than once in a list.

- It uses a dictionary called frequency to count how many times each element occurs.
- Then it checks the counts and stores elements with a count greater than one in the duplicates list.
- Finally, it returns the list of duplicate values.

Why This Version is Better

- Uses clear and descriptive names for easy understanding.
- Follows proper indentation and PEP 8 formatting.
- Includes a docstring that explains the function’s purpose, parameters, and return value.

- Structured in a way that is easy to read, maintain, and modify.

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

Expected Outcome

• Secure SQL queries using parameterized statements

• Input validation logic

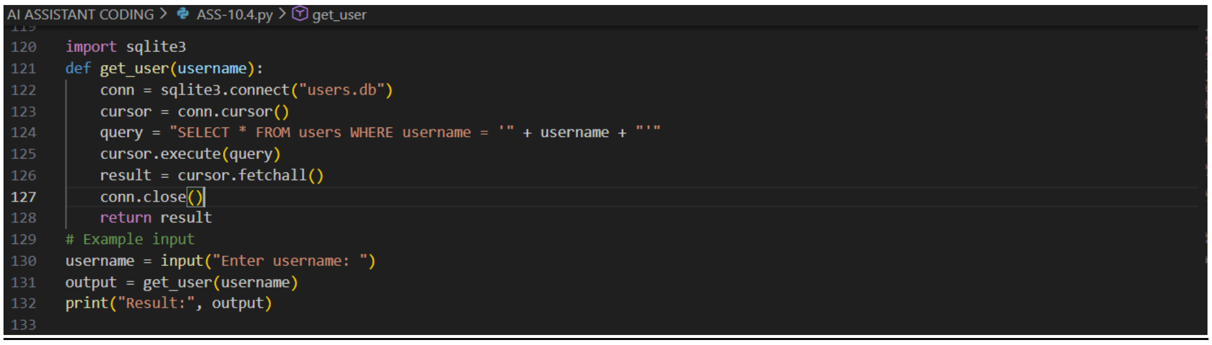
• Try-except blocks for runtime safety

• AI-generated explanation of security improvements

• Production-ready code structure

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

# Example input

username = input("Enter username: ")

output = get_user(username)

print("Result:", output)
```



```
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
# Example input
username = input("Enter username: ")
output = get_user(username)
print("Result:", output)
```

Explanation:

SQL Injection Risk: Concatenating user input directly into the query allows attackers to manipulate the database.

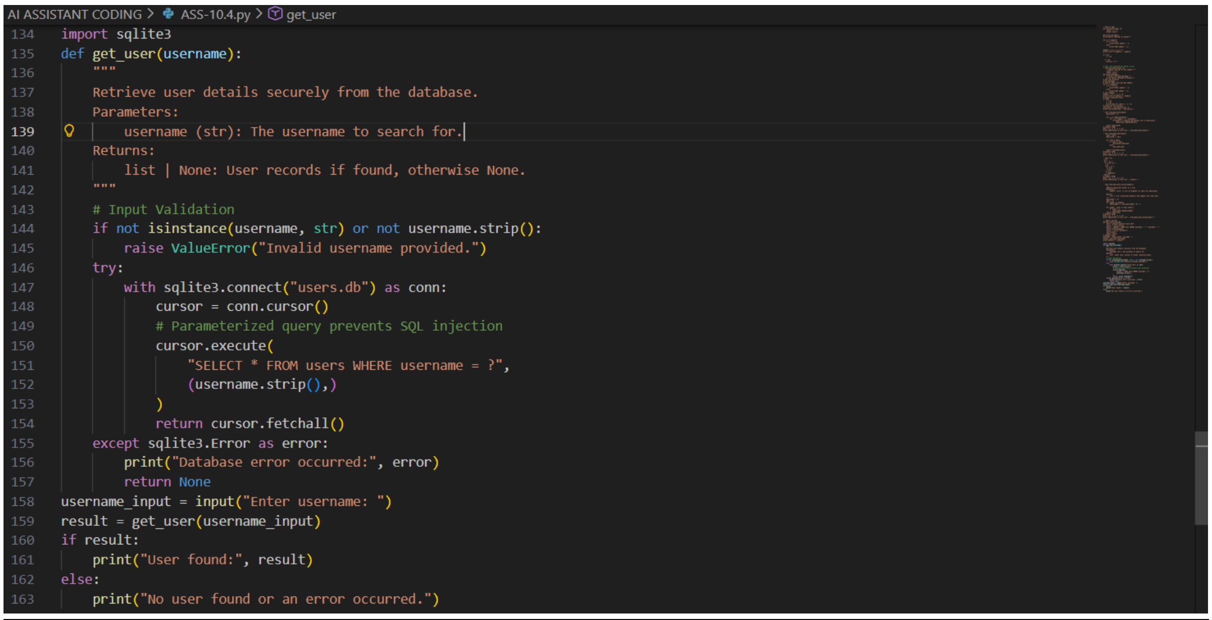
No Input Validation: Accepts any value without checking type or emptiness.

No Exception Handling: Program may crash if the database fails.

Connection Safety: Database connection may remain open during errors.

Production-Ready Secure Code with Input

```
import sqlite3

def get_user(username):

    """

    Retrieve user details securely from the database.

    Parameters:

        username (str): The username to search for.

    Returns:

        list | None: User records if found, otherwise None.

    """
    # Input Validation
    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()
            # Parameterized query prevents SQL injection
            cursor.execute(
                "SELECT * FROM users WHERE username = ?",
                (username.strip(),)
            )
            return cursor.fetchall()
    except sqlite3.Error as error:
        print("Database error occurred:", error)
        return None
username_input = input("Enter username: ")
result = get_user(username_input)
if result:
    print("User found:", result)
else:
    print("No user found or an error occurred.")
```



AI ASSISTANT CODING > ASS-10.4.py > get_user

```python
import sqlite3
def get_user(username):
    """
    Retrieve user details securely from the database.
    Parameters:
        username (str): The username to search for.
    Returns:
        list | None: User records if found, otherwise None.
    """
    # Input Validation
    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()
            # Parameterized query prevents SQL injection
            cursor.execute(
                "SELECT * FROM users WHERE username = ?",
                (username.strip(),)
            )
            return cursor.fetchall()
    except sqlite3.Error as error:
        print("Database error occurred:", error)
        return None
username_input = input("Enter username: ")
result = get_user(username_input)
if result:
    print("User found:", result)
else:
    print("No user found or an error occurred.")
```

Explanation:

Parameterized Queries: Use ? placeholders instead of string concatenation to prevent SQL injection attacks.

Input Validation: Checks that the username is valid and not empty, reducing errors.

Exception Handling: try-except blocks stop the program from crashing and handle database errors safely.

Automatic Resource Management: with sqlite3.connect() ensures the connection closes automatically.

Improved Reliability: The function handles failures gracefully and is safer for production use.

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:

o Code readability

o Naming conventions

o Formatting and style consistency

o Error handling

o Documentation quality

o Maintainability

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

issues.

Expected Outcome

• AI-generated review report including:

o Identified quality issues

o Risk areas

o Code smell detection

o Improvement suggestions

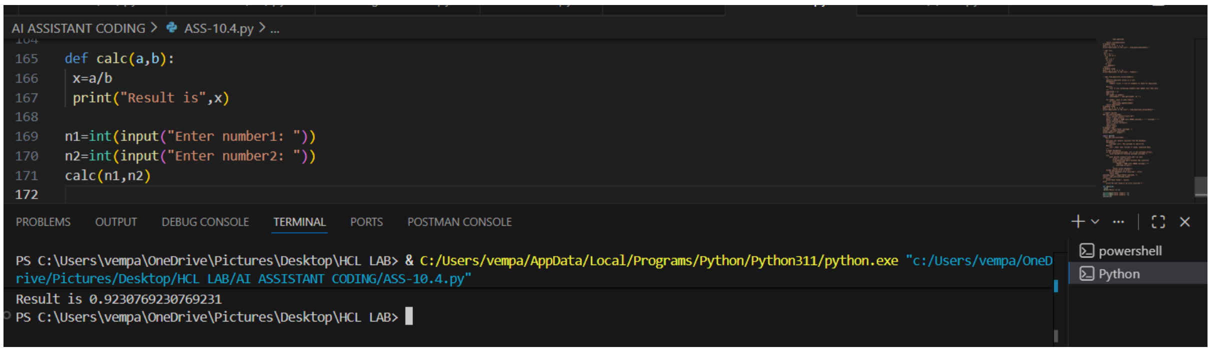
• Optional improved version of the code

• Demonstration of AI as a code reviewer, not just a code

Generator

Code:

```
def calc(a,b):
 x=a/b
 print("Result is",x)
n1=int(input("Enter number1: "))
n2=int(input("Enter number2: "))
calc(n1,n2)
```

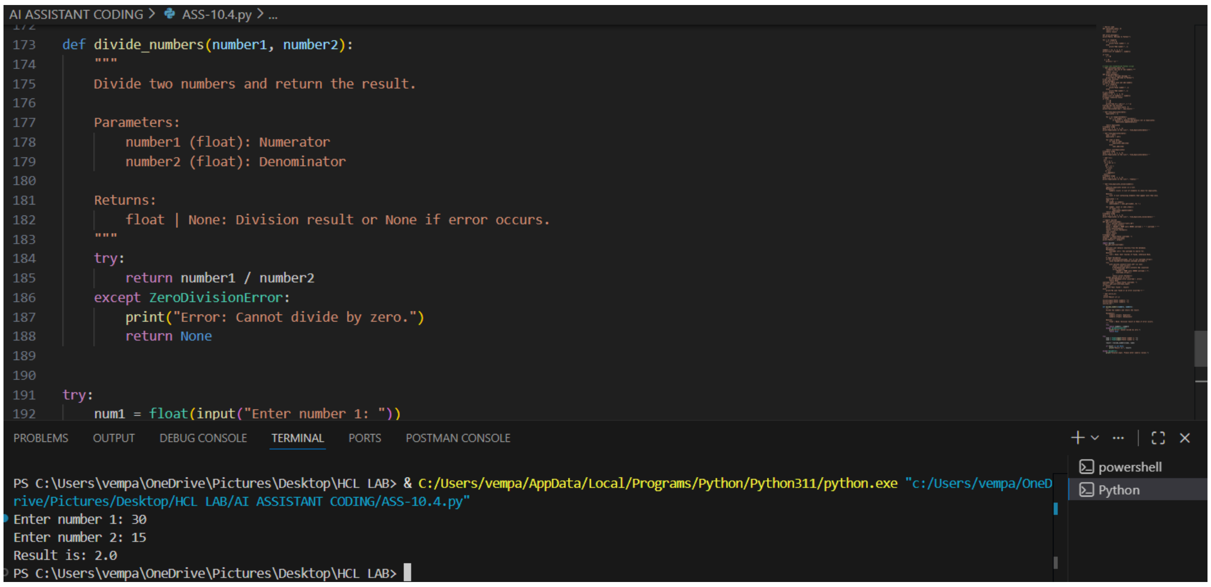


```
def calc(a,b):
 x=a/b
 print("Result is",x)

n1=int(input("Enter number1: "))
n2=int(input("Enter number2: "))
calc(n1,n2)
```

```
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
Result is 0.9230769230769231
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

Optimised Code:

```
def divide_numbers(number1, number2):
    """
    Divide two numbers and return the result.
    Parameters:
        number1 (float): Numerator
        number2 (float): Denominator
    Returns:
        float | None: Division result or None if error occurs.
    """
    try:
        return number1 / number2
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
        return None
try:
    num1 = float(input("Enter number 1: "))
    num2 = float(input("Enter number 2: "))
    result = divide_numbers(num1, num2)
    if result is not None:
        print("Result is:", result)
except ValueError:
    print("Invalid input. Please enter numeric values.")
```



AI ASSISTANT CODING > ASS-10.4.py > ...

```python
def divide_numbers(number1, number2):
    """
    Divide two numbers and return the result.

    Parameters:
        number1 (float): Numerator
        number2 (float): Denominator

    Returns:
        float | None: Division result or None if error occurs.
    """
    try:
        return number1 / number2
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
        return None


try:
    num1 = float(input("Enter number 1: "))
```

```
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vempa/OneD
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/ASS-10.4.py"
Enter number 1: 30
Enter number 2: 15
Result is: 2.0
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

Identified Quality Issues

- Poor readability and unclear structure.
- Non-descriptive function and variable names.
- No error handling or documentation.
- Formatting not aligned with PEP 8 standards.

Risk Areas

- Program may crash due to division by zero.
- Invalid user input can cause runtime errors.
- Difficult to maintain and reuse.

Code Smell Detection

- Vague naming (calc, a, b).
- Mixed logic (input and function together).
- Lack of comments/docstrings reduces clarity.

Improvement Suggestions

- Use meaningful names.
- Add try-except for error handling.
- Follow proper formatting.
- Include docstrings.
- Separate business logic from user input.

Demonstration of AI as a Code Reviewer

This review shows how AI can:

✓ Detect code smells
✓ Identify risk areas
✓ Evaluate maintainability
✓ Recommend best practices
✓ Improve code quality before human review

AI acts as a preliminary reviewer, helping teams maintain consistent and production-ready code.