Assignment-10.4

2303A51304

Batch-05

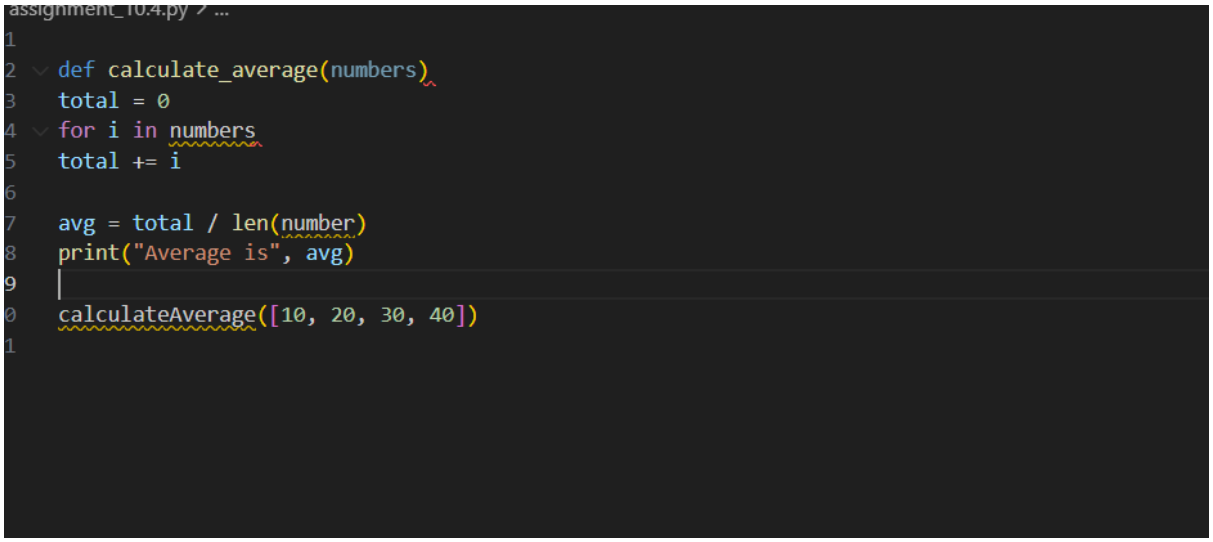
Task-01:

Prompt:

Review the following Python script. Identify and fix all syntax errors, indentation issues, incorrect variable names, and faulty function calls. Provide a corrected, executable version of the code and explain each correction made.

Code:

Wrong Code:



Corrected Code:

```
calculateAverage([10, 20, 30, 40])
'''
'''Review the following Python script. Identify and fix all syntax errors, indentation issues,
incorrect variable names, and faulty function calls.
Provide a corrected, executable version of the code and explain each correction made.'''
def calculate_average(numbers):
    total = 0
    for i in numbers:
        total += i

    avg = total / len(numbers)
    return avg


result = calculate_average([10, 20, 30, 40])
print("Average is", result)
```

Output:

Wrong Output:

```
PROBLEMS 5   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   POSTMAN CONSOLE

PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/python.exe c
:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/assignment_10.4.py
  File "c:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding\assignment_10.4.py", line 2
    def calculate_average(numbers)
                                  ^
SyntaxError: expected ':'
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Correct Output:

```
    def calculate_average(numbers)
                                  ^
SyntaxError: expected ':'
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/
:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/assignment_10.4.py
Average is 25.0
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

Syntax errors like missing colons in the function and loop were corrected.

Indentation issues were fixed to follow Python's block structure rules.

Incorrect variable names were corrected to maintain consistency.

The function name in the function call was fixed due to Python's case sensitivity.

Code structure was improved by returning values from the function and printing outside.

Task-02:

Code:

Inefficient Code:

assignment_10.4.py > ...

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


data = [1, 2, 3, 2, 4, 5, 1]
print(find_duplicates(data))
```

Output:

```
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users
Drive/Desktop/AI_Assistance_coding/assignment_10.4.py
[1, 2]
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Optimized Code:

assignment_10.4.py > ...

```python
def find_duplicates(arr):
    seen = set()
    duplicates = set()

    for value in arr:
        if value in seen:
            duplicates.add(value)
        else:
            seen.add(value)

    return list(duplicates)


data = [1, 2, 3, 2, 4, 5, 1]
print(find_duplicates(data))

```

Output:

```
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/python.ex
Drive/Desktop/AI_Assistance_coding/assignment_10.4.py
[1, 2]
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

In Inefficient code we observe:

Uses nested loops, causing unnecessary repeated comparisons

Time complexity is $O(n^2)$, which is inefficient for large lists

Checking not in duplicates adds extra overhead

Whereas in the Optimized code we observe:

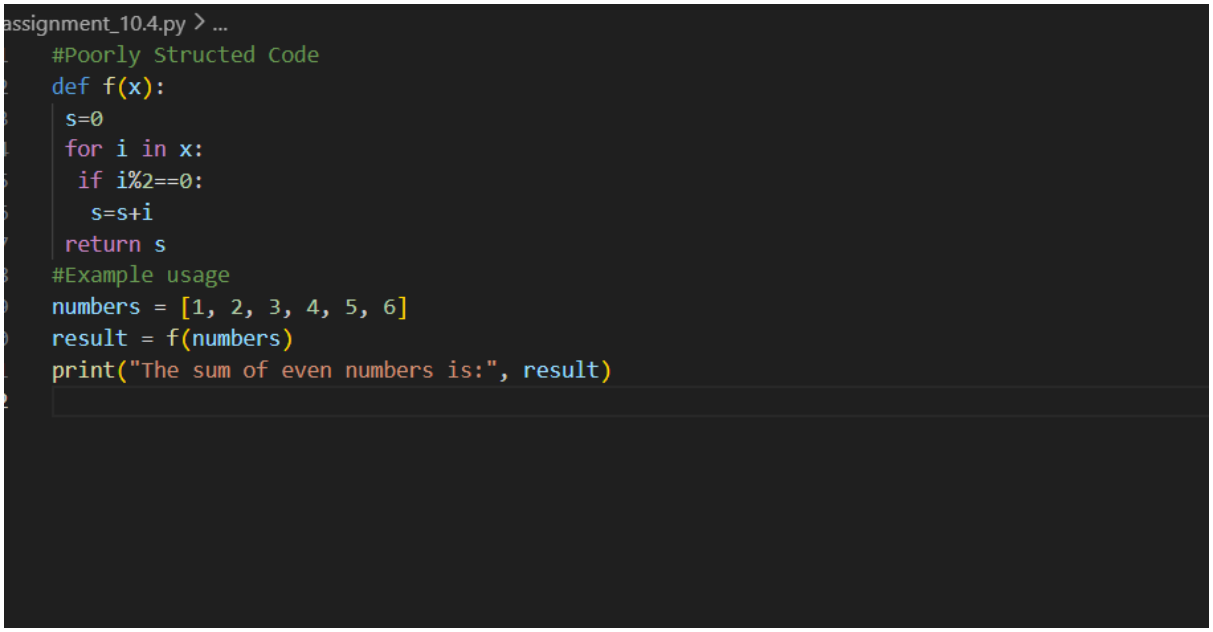
Uses sets, which provide O(1) average lookup time

Each element is processed only once

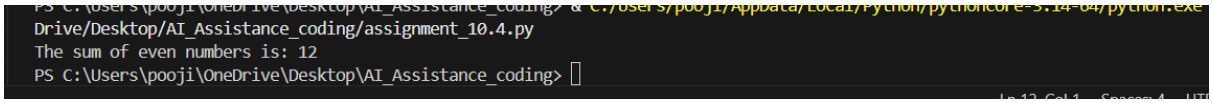
Overall time complexity improves to O(n)

Task-03:

Poorly structured code:



Output:



Well-Structured Code:

```python
#Improved Code with Proper Structure and Documentation
def calculate_sum_of_even_numbers(numbers):
    """
    Calculates the sum of all even numbers in a given list.

    Parameters:
        numbers (list): A list of integers.

    Returns:
        int: Sum of even numbers in the list.
    """
    even_sum = 0

    for number in numbers:
        if number % 2 == 0:
            even_sum += number

    return even_sum
# Example usage
if __name__ == "__main__":
    numbers = [1, 2, 3, 4, 5, 6]
    result = calculate_sum_of_even_numbers(numbers)
    print("The sum of even numbers is:", result)
```

Output:

```
The sum of even numbers is: 12
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/python.
Drive/Desktop/AI_Assistance_coding/assignment_10.4.py
The sum of even numbers is: 12
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

Problems we identify in poorly-structured code are:

Cryptic function name (f)

Unclear variable names (x, s, i)

Poor indentation and formatting

No comments or documentation

AI Explanation of Readability Improvements

1. Improved Naming Conventions

   Function name changed from f to calculate_sum_of_even_numbers

   Variables renamed to clearly reflect their purpose

2.Proper indentation (4 spaces)

   Clear spacing around operators

   Readable line structure

3. Structural Clarity

   Each logical step is clearly separated

   Code is easy to modify or extend in the future

4.Documentation Added:

   Purpose of the function

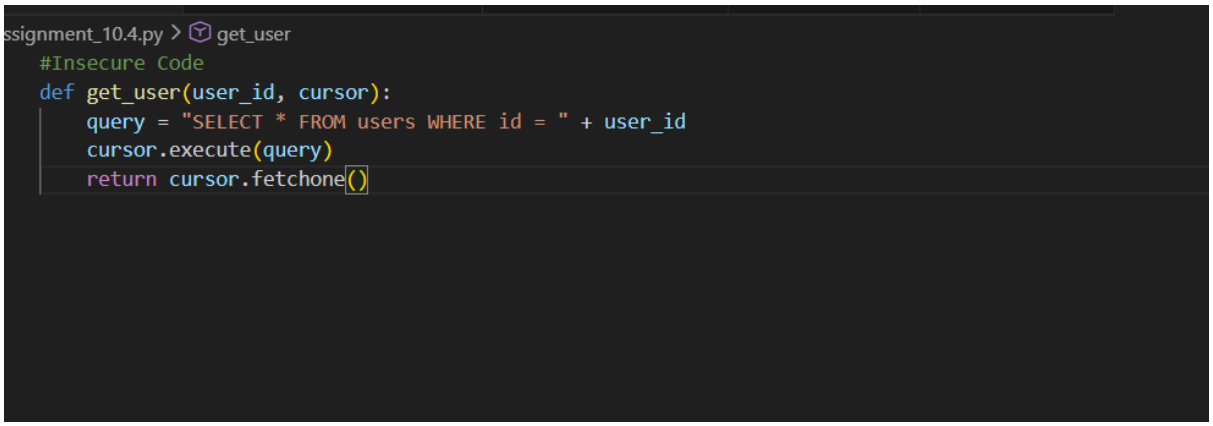
   Input parameters

   Return value

Task-04:

Code:

Insecure Code:



Secured Code:

assignment_10.4.py > ...

```python
def get_user(user_id, cursor):
    """
    Securely retrieves user details from the database using a parameterized query.

    Parameters:
        user_id (int): User ID
        cursor: Database cursor object

    Returns:
        tuple or None: User record if found, else None
    """
    try:
        # Input validation
        if not isinstance(user_id, int):
            raise ValueError("User ID must be an integer")

        # Secure parameterized query
        query = "SELECT * FROM users WHERE id = %s"
        cursor.execute(query, (user_id,))
        return cursor.fetchone()

    except ValueError as ve:
        print("Input Error:", ve)
    except Exception as e:
        print("Database Error:", e)

    return None
class MockCursor:
    def execute(self, query, params):
        print("Query executed safely:", query)
        print("Parameters:", params)

    def fetchone(self):
        return (1, "Poojitha", "poojitha@email.com")
# -------- Main Program (Testing Section) --------
if __name__ == "__main__":
    mock_cursor = MockCursor()
    user_details = get_user(1, mock_cursor)
    print("User Details:", user_details)
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
Query executed safely: SELECT * FROM users WHERE id = %s
Parameters: (1,)
User Details: (1, 'Poojitha', 'poojitha@email.com')
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

Issues in Insecure code are:

Unsafe SQL query construction (SQL Injection vulnerability)

No validation of user input

No exception handling for database errors

Application may crash or expose sensitive data

Whereas in Secured Code:

1.SQL Injection Prevention

Replaced string concatenation with **parameterized SQL query**

Prevents malicious SQL code execution

2. Input Validation

Ensured user_id is an integer before querying

Stops invalid or malicious input early

3. Exception Handling

Added try-except blocks to handle:

Input errors

Database/runtime errors

Prevents application crashes

4. Production Readiness

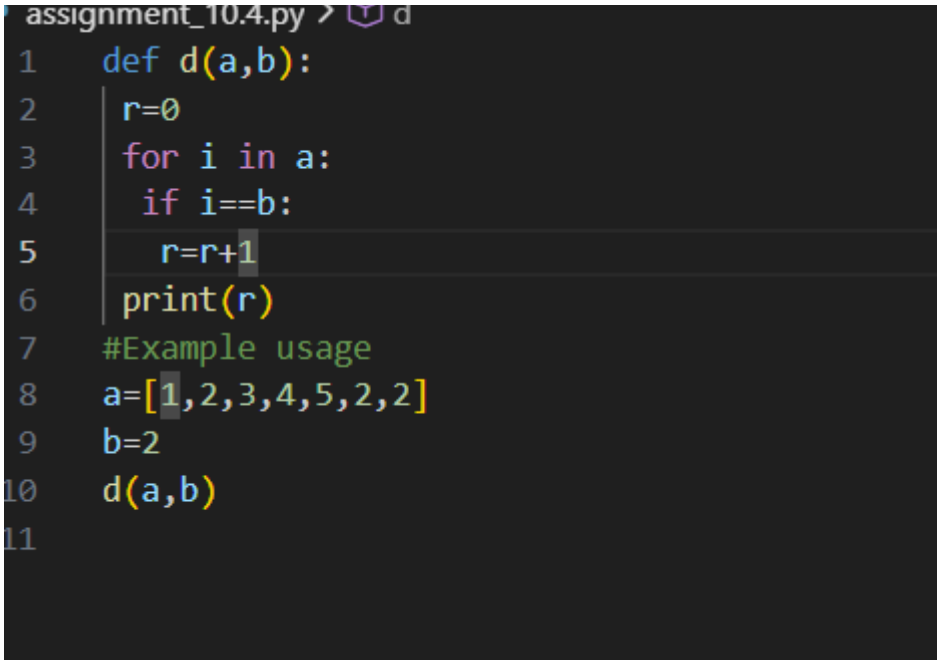
Clean structure with documentation

Safe, reliable, and maintainable backend function

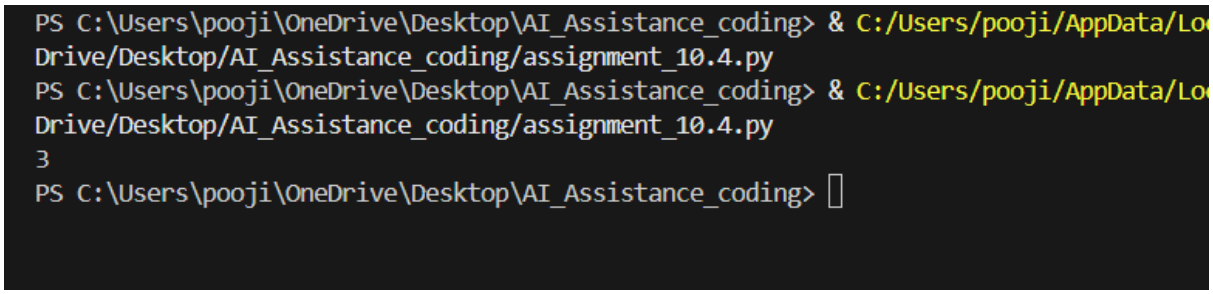
Task-05:

Code:

Poorly written python code:



Output:



Improvised Code:

```
b=2
d(a,b)
'''
#Corrected code with proper function definitions and documentation
def count_occurrences(numbers, target):
    """
    Counts how many times a target value appears in a list.

    Parameters:
        numbers (list): List of elements
        target: Value to be counted

    Returns:
        int: Number of occurrences
    """
    count = 0
    for number in numbers:
        if number == target:
            count += 1
    return count
# Example usage
if __name__ == "__main__":
    a = [1, 2, 3, 4, 5, 2, 2]
    b = 2
    result = count_occurrences(a, b)
    print(f"The number {b} appears {result} times in the list.")
```

Output:

```
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Loca
Drive/Desktop/AI_Assistance_coding/assignment_10.4.py
The number 2 appears 3 times in the list.
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

The original code uses unclear function and variable names, making it hard to understand what the code does.

Formatting and indentation in the original code are poor, which reduces readability.

The original code has no documentation, so the purpose and usage of the function are not clear.

It prints the result directly instead of returning it, which limits reusability and testing.

The improved code uses meaningful names, follows PEP 8 standards, and has a docstring for clarity.

Returning values in the improved code makes it more maintainable and suitable for larger projects.