AIAC- LAB ASSAIGNMENT 10.3

NAME: T. SAI VARUN

ROLL NO:2403A510C0

BATCH:05

Task 1: Syntax and Error Detection

Prompt:

Identify and fix syntax, indentation, and variable errors in the given script..

Code:

AIAC_10.py > ...

```python
# task1_fixed.py

def add_numbers(a: float, b: float) -> float:
    """Return the sum of two numbers."""
    result = a + b
    return result


if __name__ == "__main__":
    print(add_numbers(3, 5))  # Output: 8
    print(add_numbers(-5, 15))  # Output: 10
    print(add_numbers(2.5, 4.5))  # Output: 7.0
```

Output:

TERMINAL powershell

```
PS C:\Users\saiva\OneDrive\Desktop\AIAC> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe c:/Users/saiva/OneDrive/Desktop/AIAC/AIAC_10.py
8
10
7.0
PS C:\Users\saiva\OneDrive\Desktop\AIAC>
```

Observation:

- Added missing colon after function definition.
- Corrected variable name from 'reslt' to 'result'.
- Fixed function call with proper comma separation.
- Now the code executes and returns the correct result.

Task 2: Logical and Performance Issue Review

Prompt

Optimize inefficient logic while keeping the result correct..

Code:

AIAC_10.py > ...

```python
# task2_optimized.py

from collections import Counter
from typing import List

def find_duplicates(nums: List[int]) -> List[int]:
    """
    Return a list of values that appear more than once in `nums`.
    The returned list preserves the order of the first occurrence of each duplicated value.
    """
    counts = Counter(nums)
    duplicates = []
    seen = set()
    for num in nums:
        if counts[num] > 1 and num not in seen:
            duplicates.append(num)
            seen.add(num)
    return duplicates


if __name__ == "__main__":
    numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
    print(find_duplicates(numbers))
```

Output:

TERMINAL powershell

```
PS C:\Users\saiva\OneDrive\Desktop\AIAC> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe c:/Users/saiva/OneDrive/Desktop/AIAC/AIAC_10.py
[1, 2]
PS C:\Users\saiva\OneDrive\Desktop\AIAC>
```

Observation

- Removed nested loops, improving efficiency from $O(n^2)$ to $O(n)$.
- Used sets for faster lookups.
- Same correct result with improved scalability.

Task 3: Code Refactoring for Readability

Prompt

Refactor messy code into clean, PEP 8–compliant, well-structured code.

Code:

AIAC_10.py

```python
def calculate_factorial(n):
    """
    Calculate the factorial of a non-negative integer n.

    Args:
        n (int): Non-negative integer for which factorial is computed.

    Returns:
        int: n! (factorial of n).

    Raises:
        ValueError: If n is negative.
    """
    if n < 0:
        raise ValueError("n must be a non-negative integer.")
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result


# Example usage
print(calculate_factorial(5))  # Output: 120
```

Output:

TERMINAL

```
PS C:\Users\saiva\OneDrive\Desktop\AIAC> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe c:/Users/saiva/OneDrive/Desktop/AIAC/AIAC_10.py
120
PS C:\Users\saiva\OneDrive\Desktop\AIAC>
```

Observation

- Function renamed for clarity.
- Variables renamed for readability.
- Added proper indentation and docstring.
- PEP 8 formatting applied.

Task 4: Security and Error Handling Enhancement

Prompt

Add security practices and exception handling to the code.

Code:

task4_secure.py > ...

```python
import sqlite3

def get_user_data(user_id):
    """
    Retrieve user data from the database using a safe parameterized query.

    Args:
        user_id (str): The user ID to search for (must be numeric)

    Returns:
        list: List of tuples containing user data, or empty list if no data found

    Raises:
        ValueError: If user_id is not numeric
        sqlite3.Error: If database operation fails
    """
    # Validate input - ensure only numeric IDs are allowed
    if not user_id.isdigit():
        raise ValueError("User ID must be numeric")

    conn = None
    try:
        # Establish database connection
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()

        # Use parameterized query to prevent SQL injection
        query = "SELECT * FROM users WHERE id = ?;"
        cursor.execute(query, (user_id,))

        result = cursor.fetchall()
        return result

    except sqlite3.Error as e:
        print(f"Database error occurred: {e}")
        return []
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return []
    finally:
        # Ensure connection is always closed properly
        if conn:
            conn.close()
```

```python
def main():
    """
    Main function to handle user input and display results.
    """
    try:
        user_input = input("Enter user ID: ")
        result = get_user_data(user_input)

        if result:
            print("User data found:")
            for row in result:
                print(row)
        else:
            print("No user data found or an error occurred.")

    except ValueError as e:
        print(f"Input error: {e}")
    except KeyboardInterrupt:
        print("\nOperation cancelled by user.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    main()
```

Output:

```
TERMINAL                                                    powershell

PS C:\Users\saiva\OneDrive\Desktop\AIAC> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\
python.exe c:/Users/saiva/OneDrive/Desktop/AIAC/AIAC_10.py
Enter user ID (integer): 1
User found: ('Alice', 'alice@example.com')
PS C:\Users\saiva\OneDrive\Desktop\AIAC>
```

Observation

- Prevented SQL injection with parameterized query.
- Added error handling with try-except.
- Added input validation for safer execution.

Task 5: Automated Code Review Report Generation

Prompt

Generate a review report for this messy code.

Code:

AIAC_10.py > ...

```python
# task5_refactored.py

from typing import Union

Number = Union[int, float]

def calculate(x: Number, y: Number, operation: str) -> Number:
    """
    Perform a basic arithmetic operation on x and y.

    Supported operations (case-insensitive):
        - "add": addition (x + y)
        - "sub": subtraction (x - y)
        - "mul": multiplication (x * y)
        - "div": division (x / y) -- raises ZeroDivisionError if y == 0

    Args:
        x (int | float): left operand
        y (int | float): right operand
        operation (str): operation name ("add", "sub", "mul", "div")

    Returns:
        int | float: result of the operation

    Raises:
        ValueError: if the operation is not supported.
        ZeroDivisionError: if operation is "div" and y == 0.
    """
    op = operation.strip().lower()
    if op == "add":
        return x + y
    elif op == "sub":
        return x - y
    elif op == "mul":
        return x * y
    elif op == "div":
        if y == 0:
            raise ZeroDivisionError("Division by zero is not allowed.")
        return x / y
    else:
        raise ValueError(f"Unsupported operation: {operation!r}. Supported: add, sub, mul, div.")

```

```python
if __name__ == "__main__":
    # Demonstration and safe usage
    try:
        print(calculate(10, 5, "add"))  # 15
    except Exception as e:
        print("Error in first calculation:", e)

    try:
        print(calculate(10, 0, "div"))
    except ZeroDivisionError as zde:
        print("Caught a division error:", zde)
    except Exception as e:
        print("Other error:", e)
```

Output

```
TERMINAL                                                    powershell
PS C:\Users\saiva\OneDrive\Desktop\AIAC> & C:\Users\saiva\AppData\Local\Programs\Python\Python31
python.exe c:/Users/saiva/OneDrive/Desktop/AIAC/AIAC_10.py
15
Caught a division error: Division by zero is not allowed.
PS C:\Users\saiva\OneDrive\Desktop\AIAC>
```

Observation

Code Review Report:
- Function and variable names are non-descriptive.
- No docstrings provided.
- Inconsistent formatting and indentation.
- No error handling for division by zero.
- Suggest renaming to descriptive names, adding docstrings, using try-except, and applying PEP 8 formatting.