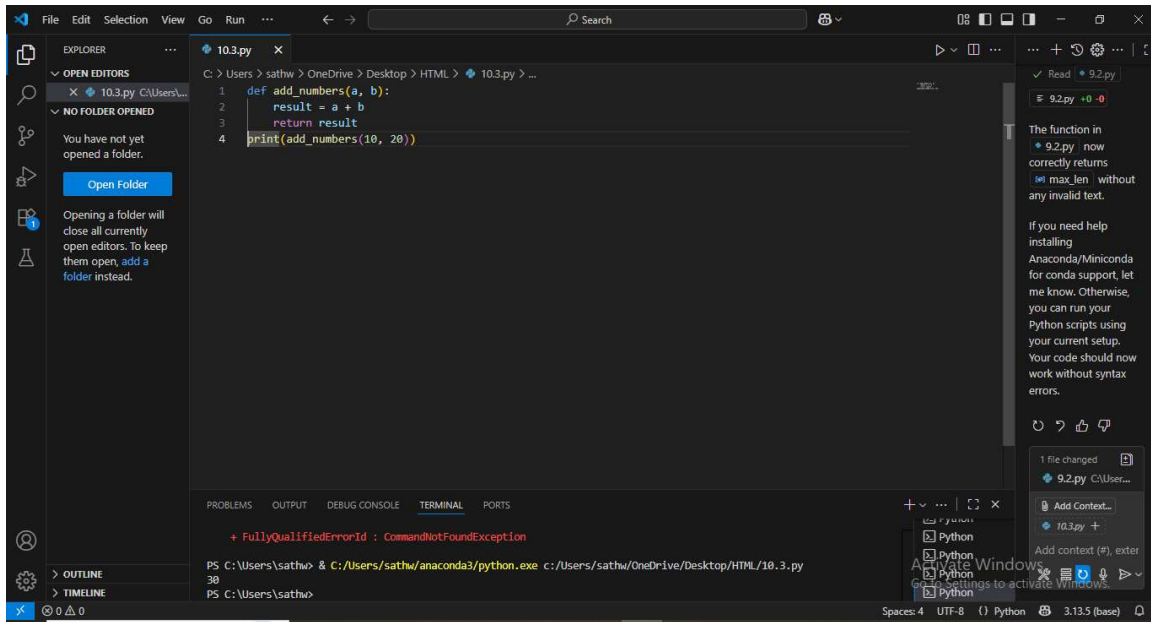


ASSIGNMENT-10.4

2403A52060

BATCH-03

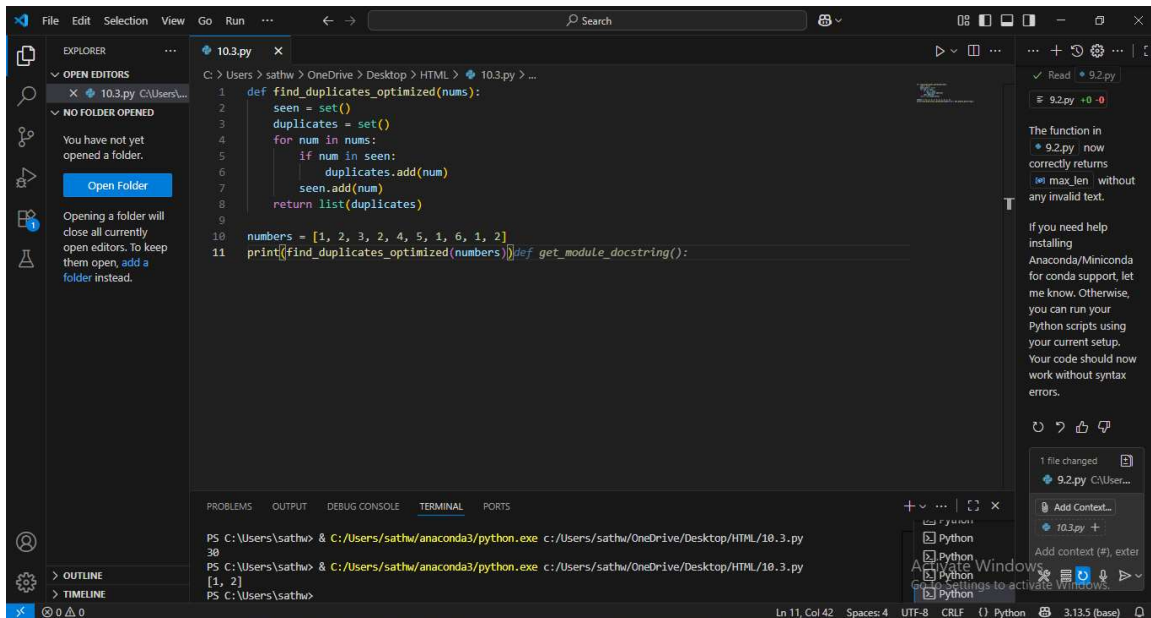
TASK-01 SYNTAX AND ERROR DETECTION



EXPLANATION:

Error in code Task 1: The original prompt contained several errors in the Python script: Syntax Error: The `def add_numbers(a, b)` line was missing a colon `:` at the end. Indentation Error: The lines `result = a + b` and `return result` were not indented correctly within the `add_numbers` function. Python uses indentation to define code blocks. Variable Error: The line `return result` had a typo; the variable `result` was used instead of `reslt`. Syntax Error: The `print(add_numbers(10 20))` line was missing a comma `,` between the arguments `10` and `20`.

TASK-02 LOGICAL AND PERFORMANCE ISSUE REVIEW



The screenshot shows a Visual Studio Code editor window. The Explorer panel on the left shows a file named '10.3.py' in the directory 'C:\Users\sathw\OneDrive\Desktop\HTML'. The main editor area displays the following Python code:

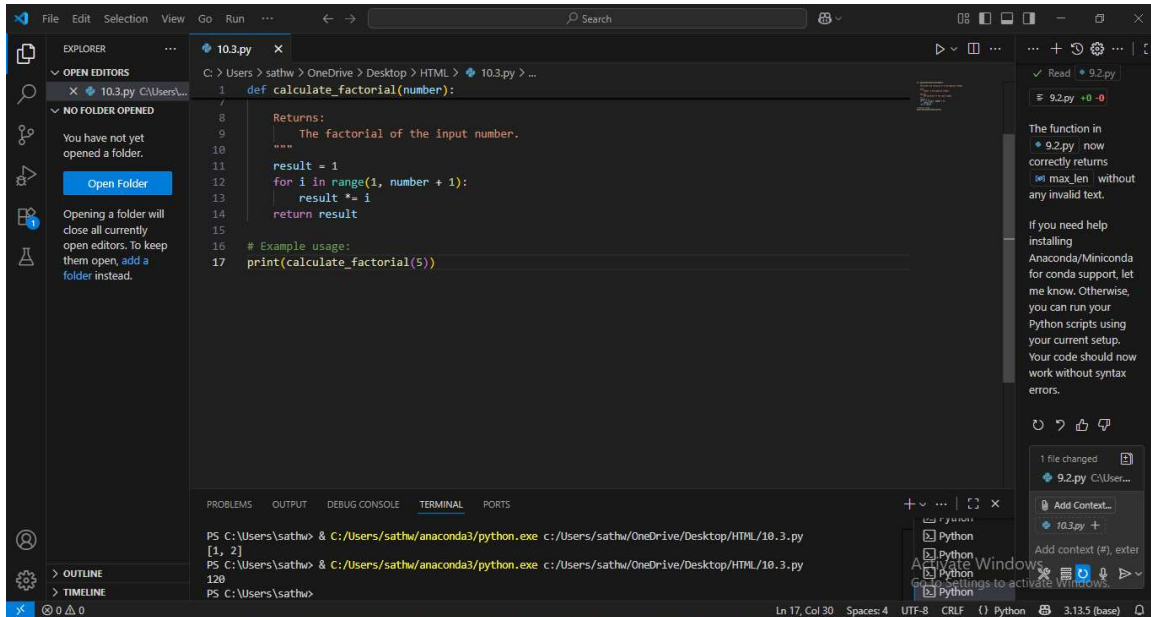
```
1 def find_duplicates_optimized(nums):
2     seen = set()
3     duplicates = set()
4     for num in nums:
5         if num in seen:
6             duplicates.add(num)
7         seen.add(num)
8     return list(duplicates)
9
10 numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
11 print(find_duplicates_optimized(numbers))
```

The output of the script is shown in the terminal at the bottom: `PS C:\Users\sathw> c:\Users\sathw\anaconda3\python.exe c:\Users\sathw\OneDrive\Desktop\HTML\10.3.py` followed by `30` and `[1, 2]`. The status bar at the bottom indicates 'Ln 11, Col 42', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Python 3.13.5 (base)'.

EXPLANATION:

Instead of using nested loops to compare every element with every other element (which is inefficient, especially for large lists), I used a more efficient approach with sets: initialized two empty sets: `seen` to keep track of the numbers encountered so far, and `duplicates` to store the numbers that appear more than once. I iterated through the input list `nums` only once. For each `num` in the list: I checked if `num` is already in the `seen` set. If it is, it means we've encountered this number before, so it's a duplicate. I added it to the `duplicates` set. I then added the current `num` to the `seen` set. Finally, I converted the `duplicates` set back to a list before returning it. This approach has a time complexity of $O(n)$, where n is the number of elements in the list, which is much more efficient than the original $O(n^2)$ approach, especially for larger datasets.

TASK-03 CODE REFACTORING FOR READABILITY



EXPLANATION:

I refactored the code to make it more readable and follow PEP 8 guidelines. Here are the specific changes I made:

- Function Name:** I changed the function name from the very short and uninformative `c` to a more descriptive name, `calculate_factorial`. This makes it clear what the function does.
- Variable Names:** I changed the variable names `n` to `number` and `x` to `result` to improve readability.
- Formatting:** I added proper indentation and spacing to make the code easier to read and understand, adhering to PEP 8 standards.
- Docstring:** I added a docstring to the `calculate_factorial` function. A docstring explains what the function does, its arguments (Args), and what it returns (Returns). This is a crucial part of writing well-documented and maintainable code.
- Example Usage:** I added a comment to clarify the example usage of the function.

TASK-04 SECURITY AND ERROR HANDLING ENHANCEMENT

VS Code editor showing the first part of a Python script (10.3.py) that defines a function to retrieve user data securely from a SQLite database. The function includes input validation and a parameterized query to prevent SQL injection.

```
1 import sqlite3
2
3 def get_user_data_secure(user_id):
4     """
5     Retrieves user data from the database securely.
6
7     Args:
8         user_id: The ID of the user to retrieve.
9
10    Returns:
11        The user data or None if an error occurs or no user is found.
12    """
13    conn = None # Initialize connection to None
14    try:
15        # Input validation: Ensure user_id is a digit
16        if not user_id.isdigit():
17            print("Error: Invalid user ID. Please enter a number.")
18            return None
19
20        conn = sqlite3.connect("users.db")
21        cursor = conn.cursor()
22
23        # Use parameterized query to prevent SQL injection
24        query = "SELECT * FROM users WHERE id = ?;"
25        cursor.execute(query, (user_id,))
```

Terminal output:

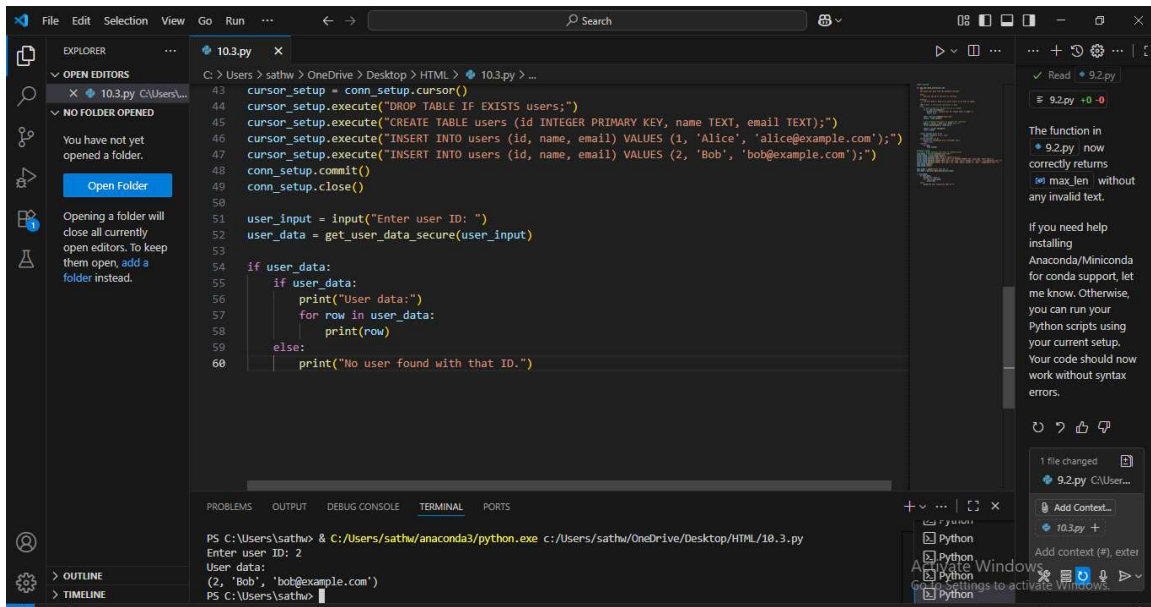
```
PS C:\Users\sathw> & C:\Users\sathw\anaconda3\python.exe c:\Users\sathw\OneDrive\Desktop\HTML\10.3.py
Enter user ID: 2
User data:
(2, 'Bob', 'bob@example.com')
```

VS Code editor showing the second part of the Python script (10.3.py) which includes exception handling, database setup, and example usage. The script demonstrates how to use the `get_user_data_secure` function.

```
26         result = cursor.fetchall()
27         return result
28
29
30 except sqlite3.Error as e:
31     print(f"Database error: {e}")
32     return None
33 except Exception as e:
34     print(f"An unexpected error occurred: {e}")
35     return None
36 finally:
37     if conn:
38         conn.close()
39
40 # Example usage:
41 # Create a dummy database and table for demonstration
42 conn_setup = sqlite3.connect("users.db")
43 cursor_setup = conn_setup.cursor()
44 cursor_setup.execute("DROP TABLE IF EXISTS users;")
45 cursor_setup.execute("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT, email TEXT);")
46 cursor_setup.execute("INSERT INTO users (id, name, email) VALUES (1, 'Alice', 'alice@example.com');")
47 cursor_setup.execute("INSERT INTO users (id, name, email) VALUES (2, 'Bob', 'bob@example.com');")
48 conn_setup.commit()
49 conn_setup.close()
```

Terminal output:

```
PS C:\Users\sathw> & C:\Users\sathw\anaconda3\python.exe c:\Users\sathw\OneDrive\Desktop\HTML\10.3.py
Enter user ID: 2
User data:
(2, 'Bob', 'bob@example.com')
```



```
43 cursor_setup = conn_setup.cursor()
44 cursor_setup.execute("DROP TABLE IF EXISTS users;")
45 cursor_setup.execute("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT, email TEXT);")
46 cursor_setup.execute("INSERT INTO users (id, name, email) VALUES (1, 'Alice', 'alice@example.com');")
47 cursor_setup.execute("INSERT INTO users (id, name, email) VALUES (2, 'Bob', 'bob@example.com');")
48 conn_setup.commit()
49 conn_setup.close()

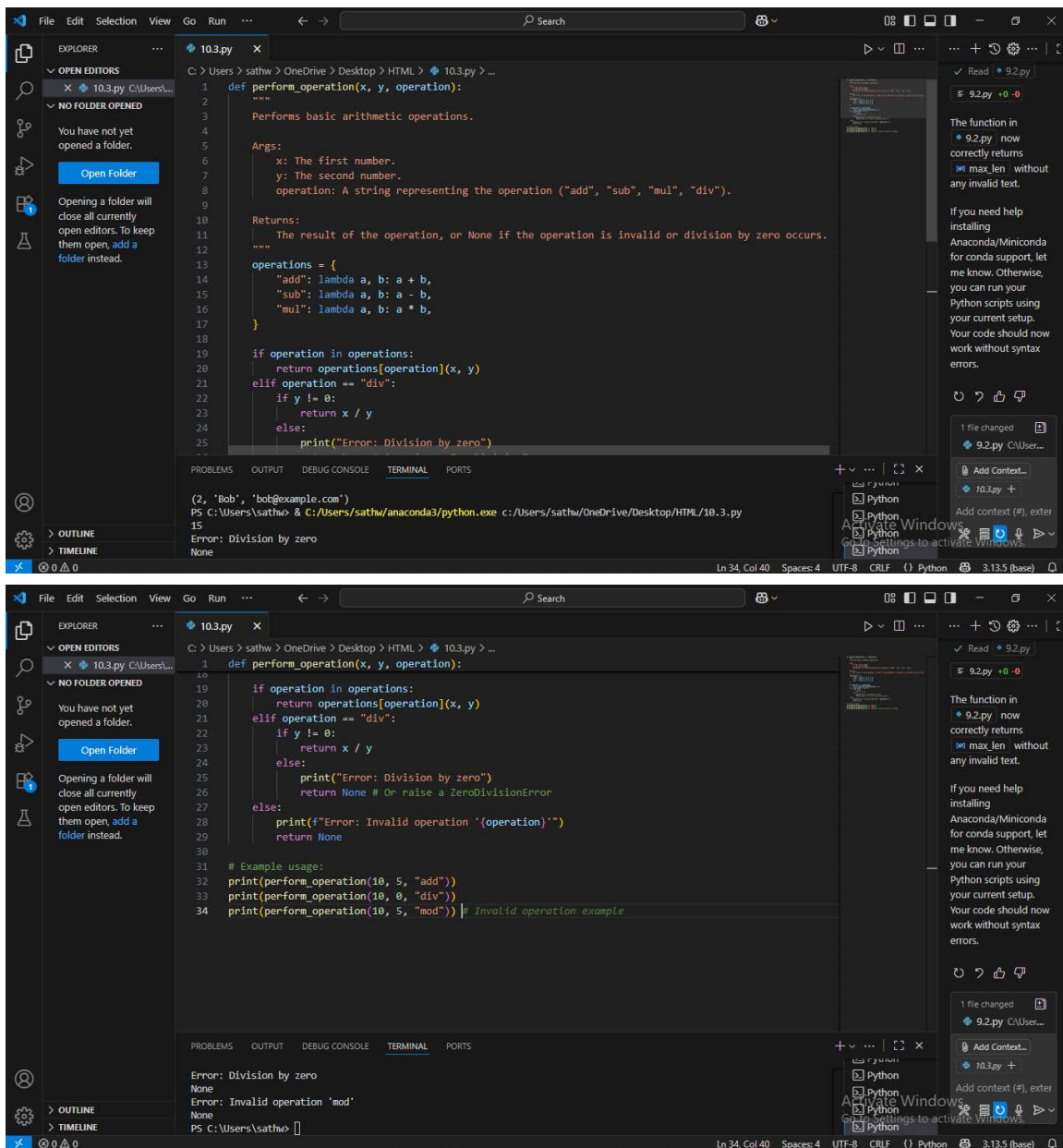
50
51 user_input = input("Enter user ID: ")
52 user_data = get_user_data_secure(user_input)
53
54 if user_data:
55     if user_data:
56         print("User data:")
57         for row in user_data:
58             print(row)
59     else:
60         print("No user found with that ID.")
```

PS C:\Users\sathw> & C:\Users\sathw\anaconda3\python.exe c:\Users\sathw\OneDrive\Desktop\HTML\10.3.py
Enter user ID: 2
User data:
(2, 'Bob', 'bob@example.com')

EXPLANATION:

Using parameterized queries (? placeholder) to prevent SQL injection instead of putting the user ID directly in the query string.
Adding input validation to check if the user ID is a number.
Including try...except...finally blocks to handle database errors and ensure the connection is always closed.

TASK-05 AUTOMATED CODE REVIEW REPORT GENERATION



EXPLANATION:

The original code defines a function `calc` to do simple math (+, -, *, /) based on text input. It prints "wrong" for invalid operations but crashes on division by zero. The improved code, `perform_operation`, does the same math but is better because: It uses clearer names and formatting (PEP 8). It handles division by zero gracefully, returning `None`. It handles invalid operations by printing an error and returning `None`. It's structured using a dictionary, making it easier

to add more operations later.