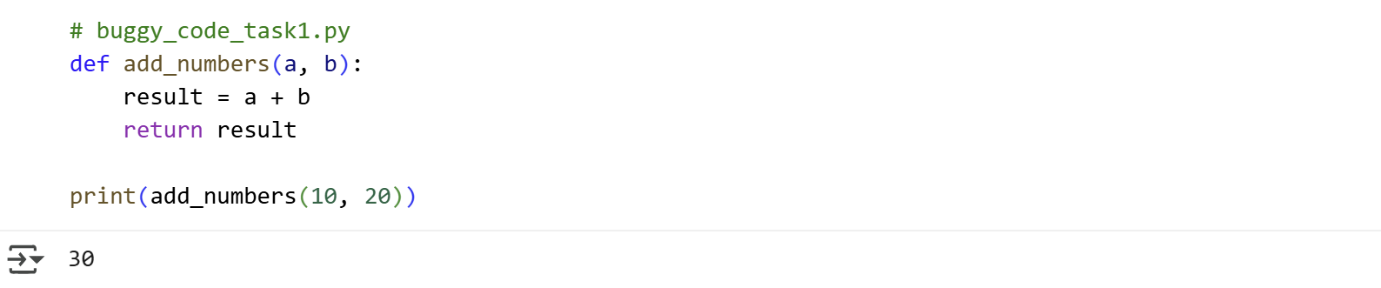
HTNO:2403A52134

Assignment-10

Task-1:

Code & output:

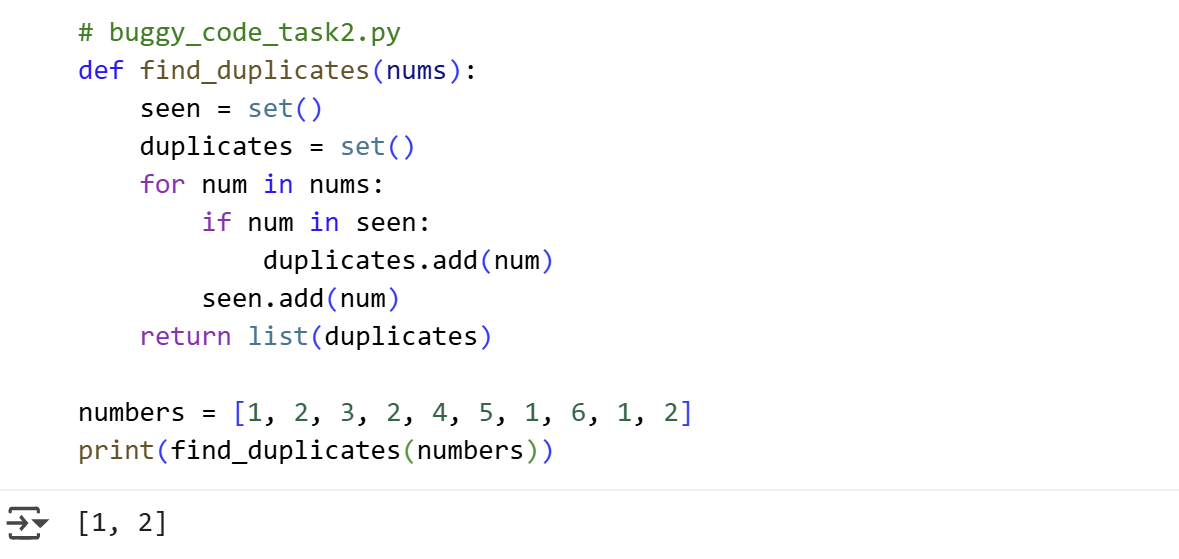


Explanation:

* def add\_numbers(a, b):: This line defines a function named add\_numbers that accepts two parameters, a and b.
* result = a + b: Inside the function, this line adds the values of a and b and assigns the sum to a variable named result.
* return result: This line returns the value stored in the result variable from the function.
* print(add\_numbers(10, 20)): This line calls the add\_numbers function with the arguments 10 and 20. The value returned by the function (which is 30) is then printed to the console.

Task-2:

Code & Output:

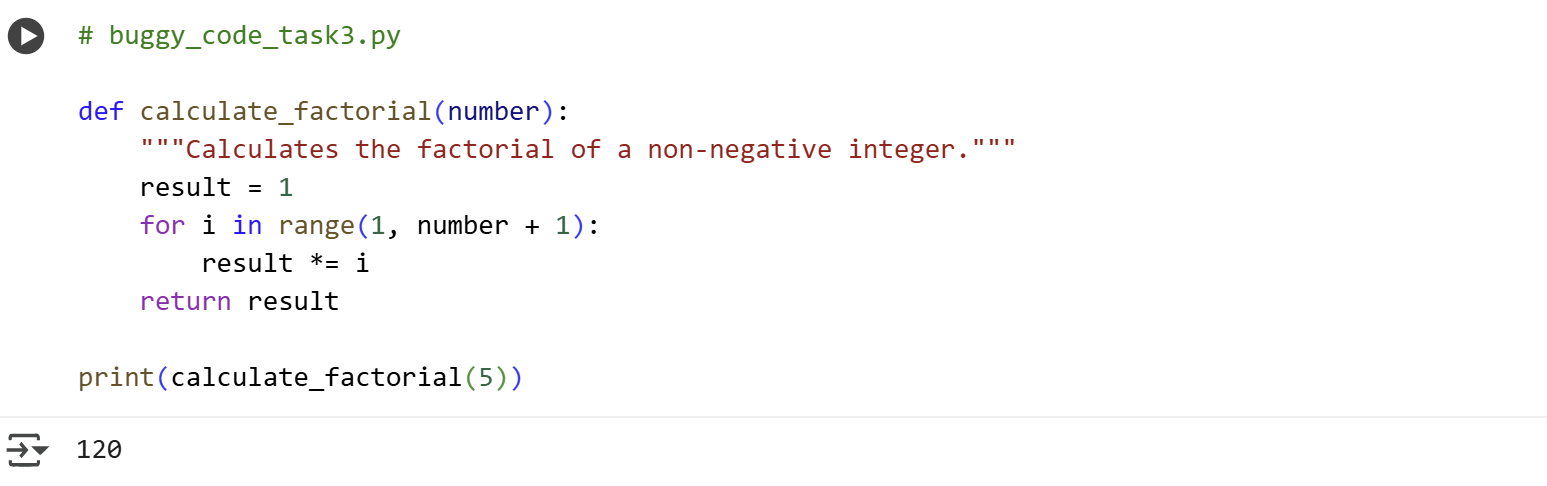


Explanation:

* def find\_duplicates(nums):: This line defines a function named find\_duplicates that takes one argument, a list of numbers named nums.
* seen = set(): This line initializes an empty set called seen. This set will be used to keep track of the numbers encountered so far in the input list.
* duplicates = set(): This line initializes an empty set called duplicates. This set will store the numbers that are found to be duplicates. Using a set automatically handles uniqueness, so each duplicate number will only be added once.
* for num in nums:: This line starts a for loop that iterates through each element in the input list nums, assigning the current element to the variable num in each iteration.
* if num in seen:: Inside the loop, this line checks if the current number num is already present in the seen set. If it is, it means this is a duplicate number.
* duplicates.add(num): If the if condition is true (the number is a duplicate), this line adds the duplicate number num to the duplicates set.
* seen.add(num): This line adds the current number num to the seen set, regardless of whether it was a duplicate or not. This marks the number as having been encountered.
* return list(duplicates): After the loop finishes, this line converts the duplicates set into a list and returns it. The list contains all the unique duplicate numbers found in the original list.
* numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]: This line creates a list of integers and assigns it to the variable numbers. This is the input data for the function.
* print(find\_duplicates(numbers)): This line calls the find\_duplicates function with the numbers list as input and prints the returned list of duplicates to the console.

Task-3:

Code&Output:

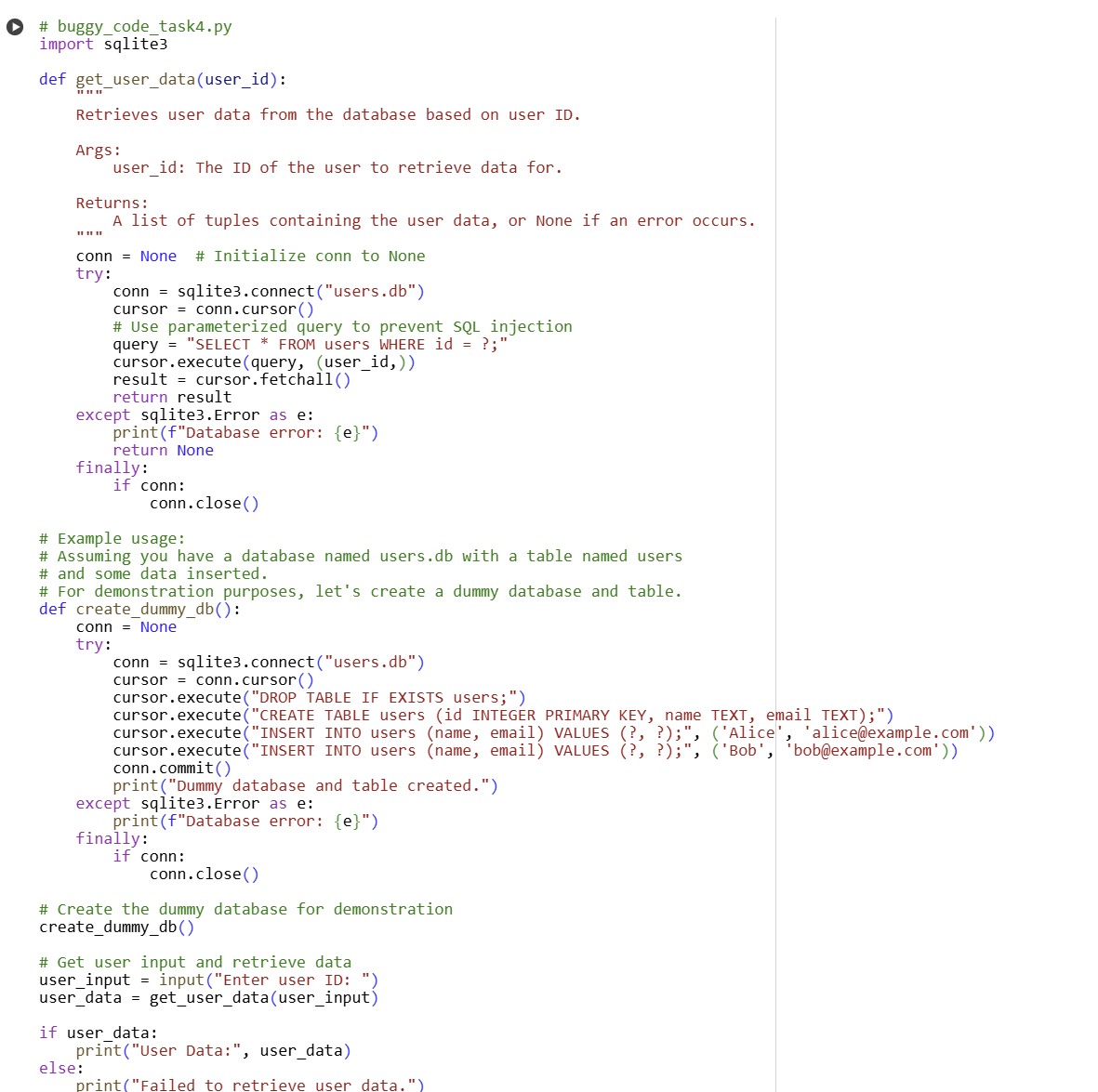


Explanation:

* # buggy\_code\_task3.py: This is a comment indicating the name of the file the code might have originated from.
* def calculate\_factorial(number):: This line defines a function named calculate\_factorial that takes one argument, an integer named number. This function is designed to calculate the factorial of this number.
* """Calculates the factorial of a non-negative integer.""": This is a docstring, which provides a brief explanation of what the function does.
* result = 1: This line initializes a variable called result and sets its initial value to 1. This is the starting point for calculating the factorial.
* for i in range(1, number + 1):: This line starts a for loop. The range(1, number + 1) function generates a sequence of numbers starting from 1 and going up to (and including) the value of number. The loop will iterate through each number in this sequence, assigning the current number to the variable i.
* result \*= i: Inside the loop, this line multiplies the current value of result by the current value of i and updates result with the new value. This is the core of the factorial calculation, where the result is successively multiplied by each integer from 1 to number.
* return result: After the loop finishes, this line returns the final value stored in the result variable, which is the calculated factorial of the input number.
* print(calculate\_factorial(5)): This line calls the calculate\_factorial function with the argument 5. The value returned by the function (which is 120, the factorial of 5) is then printed to the console.

Task-4:

Code:



Output:

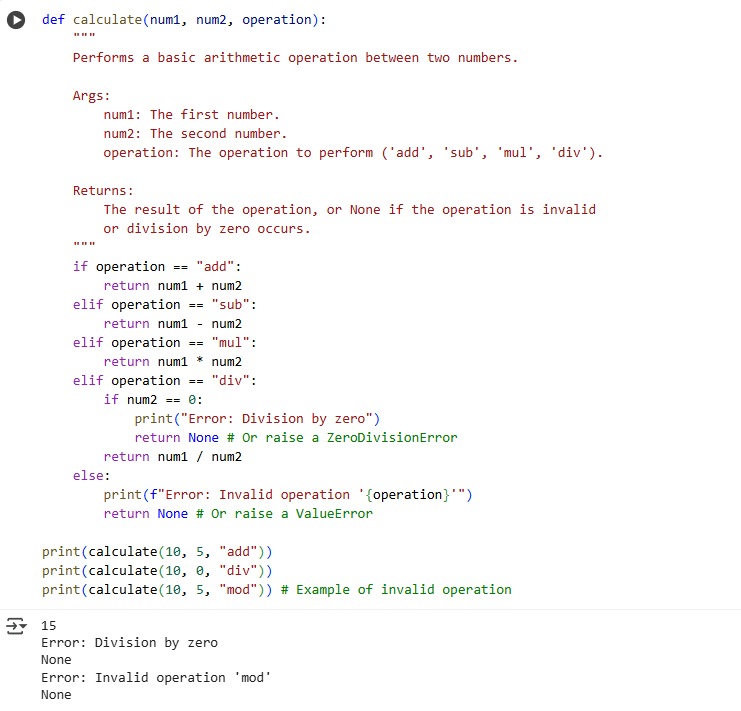


Explanation:

* # buggy\_code\_task4.py: This is a comment indicating the name of the file the code might have originated from.
* import sqlite3: This line imports the sqlite3 module, which provides an interface for working with SQLite databases in Python.
* def get\_user\_data(user\_id):: This line defines a function named get\_user\_data that takes one argument, user\_id, which is intended to be the ID of the user to retrieve data for.
* """...""": This is a docstring explaining the purpose of the get\_user\_data function, its arguments, and what it returns.
* conn = None # Initialize conn to None: This line initializes a variable conn to None. This variable will hold the database connection object. It's initialized to None so that it can be checked in the finally block.
* try:: This line starts a try block, which is used for exception handling. Code within this block will be monitored for potential errors.
* conn = sqlite3.connect("users.db"): This line establishes a connection to the SQLite database file named "users.db". If the file doesn't exist, it will be created. The connection object is assigned to the conn variable.
* cursor = conn.cursor(): This line creates a cursor object from the connection. A cursor is used to execute SQL commands.
* # Use parameterized query to prevent SQL injection: This is a comment explaining the purpose of the next line.
* query = "SELECT \* FROM users WHERE id = ?;": This line defines the SQL query to select all columns (\*) from the "users" table where the "id" column matches a specific value. The ? is a placeholder for a parameter, which is a security measure against SQL injection.
* cursor.execute(query, (user\_id,)): This line executes the SQL query. The user\_id is passed as a parameter in a tuple (user\_id,). SQLite will safely substitute the user\_id into the query, preventing malicious input from affecting the SQL command.
* result = cursor.fetchall(): This line fetches all the rows returned by the executed query and stores them as a list of tuples in the result variable.
* return result: This line returns the result (the retrieved user data) from the function.
* except sqlite3.Error as e:: This line starts an except block that catches any sqlite3.Error that might occur within the try block.
* print(f"Database error: {e}"): If a database error occurs, this line prints an error message including the specific error details.
* return None: If a database error occurs, this line returns None to indicate that the data retrieval failed.
* finally:: This line starts a finally block, which contains code that will be executed regardless of whether an exception occurred or not.
* if conn:: This line checks if the conn variable is not None (meaning a database connection was successfully established).
* conn.close(): If a connection exists, this line closes the database connection to release resources.
* # Example usage:: This is a comment indicating the start of example usage code.
* def create\_dummy\_db():: This line defines a function named create\_dummy\_db to create a sample database and table for demonstration.
* conn = None: Initializes conn to None within the create\_dummy\_db function.
* try:: Starts a try block for error handling within create\_dummy\_db.
* conn = sqlite3.connect("users.db"): Connects to the "users.db" database within create\_dummy\_db.
* cursor = conn.cursor(): Creates a cursor for create\_dummy\_db.
* cursor.execute("DROP TABLE IF EXISTS users;"): Executes an SQL command to drop the "users" table if it already exists.
* cursor.execute("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT, email TEXT);"): Executes an SQL command to create a new "users" table with columns for id, name, and email.
* cursor.execute("INSERT INTO users (name, email) VALUES (?, ?);", ('Alice', 'alice@example.com')): Inserts a row with 'Alice' into the "users" table using parameterized query.
* cursor.execute("INSERT INTO users (name, email) VALUES (?, ?);", ('Bob', 'bob@example.com')): Inserts a row with 'Bob' into the "users" table using parameterized query.
* conn.commit(): Commits the changes made to the database within create\_dummy\_db.
* print("Dummy database and table created."): Prints a confirmation message.
* except sqlite3.Error as e:: Catches database errors in create\_dummy\_db.
* print(f"Database error: {e}"): Prints error message for create\_dummy\_db.
* finally:: finally block for create\_dummy\_db.
* if conn:: Checks if connection exists in create\_dummy\_db.
* conn.close(): Closes the connection in create\_dummy\_db.
* create\_dummy\_db(): Calls the create\_dummy\_db function to set up the database.
* user\_input = input("Enter user ID: "): Prompts the user to enter a user ID and stores the input in the user\_input variable.
* user\_data = get\_user\_data(user\_input): Calls the get\_user\_data function with the user's input and stores the returned data in the user\_data variable.
* if user\_data:: Checks if user\_data is not None (meaning data was successfully retrieved).
* print("User Data:", user\_data): If data was retrieved, this line prints the retrieved user data.
* else:: If user\_data is None (meaning data retrieval failed or no user with that ID was found).
* print("Failed to retrieve user data."): This line prints a message indicating that data retrieval failed.

Task-5:

Code & Output:



Explanation:

