Explaining sections of code

```
import easygui as eg
import sqlite3

# --- Database Setup and Functions ---
def setup_database():
    """
    Connects to the SQLite database and ensures the contacts table exists.
    If the database file doesn't exist, SQLite will create it.
    This function returns the connection and cursor objects for later use.
    """
    try:
        # Connect to the database file named 'contacts.db'.
        # This will create the file if it doesn't exist.
        conn = sqlite3.connect('relational_database.db')

        # Create a cursor object to execute SQL commands.
        cursor = conn.cursor()

        # SQL command to create the 'contacts' table if it doesn't already exist.
        # This prevents an error if you run the script multiple times.
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Customers (
                customer_id INTEGER PRIMARY KEY,
                first_name TEXT NOT NULL,
                last_name TEXT
            )
        ''')

        # Another table is created
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Orders (
                order_id INTEGER PRIMARY KEY,
                customer_id INTEGER NOT NULL,
                order_date TEXT NOT NULL,
                total_amount REAL NOT NULL,
                FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
            )
        ''')

        # Commit the changes to save the table creation to the database file.
        conn.commit()

        # Return the connection and cursor for use in other functions.
        return conn, cursor

    except sqlite3.Error as e:
        # Use EasyGui to show an error message if the database connection fails.
        eg.exceptionbox(msg=f"A database error occurred: {e}", title="Database Error")
        # Return None to signal that a fatal error occurred.
        return None, None


        # Use EasyGui to show an error message if the database connection fails.
        eg.exceptionbox(msg=f"A database error occurred: {e}", title="Database Error")
        # Return None to signal that a fatal error occurred.
        return None, None
```

This block of code sets up the entire database system for the program. It begins by connecting to a SQLite database file called *relational_database.db*, creating the file automatically if it does not already exist. A cursor is then created so the program can run SQL commands. The code checks for two tables, Customers and Orders, and creates them if they are missing. This allows the database to store customer information and their related orders. The function then commits these changes so they are saved permanently and returns both the database connection and cursor so other parts of the program can interact with the database. If anything goes wrong during this process, an EasyGUI error box appears to tell the user what happened, and the function safely returns None to stop the program from continuing.

```
def add_customer(conn, cursor):
    """
    Prompts the user for a new customer's details and inserts them into the
Customers table.
    """
    msg = "Enter new customer information"
    title = "Add Customer"
    fieldNames = ["customer_id", "first_name", "last_name"]

    fieldValues = eg.multenterbox(msg, title, fieldNames)

    if fieldValues is None:
        return

    customer_id, first_name, last_name = fieldValues
    # Checks if any input is empty; shows error and stops if so
    if not customer_id or not first_name or not last_name:
        eg.msgbox("All boxes need to be filled!", "Input Erroe")
        return

    # Tries to add a new row to the Customers table with the provided values
    try:
        cursor.execute("INSERT INTO Customers VALUES (?, ?, ?)",
            (customer_id, first_name, last_name))

        conn.commit()
        eg.msgbox(f"Customer '{first_name}' added successfully!", "Success")

    except sqlite3.IntegrityError:
        eg.msgbox(f"Error: Customer ID '{customer_id}' already exists.", "Database
Error")

    except sqlite3.Error as e:
        eg.exceptionbox(msg=f"Failed to add customer: {e}", title="Database Error")
```

This function allows the user to add a new customer to the database. It first displays an EasyGUI input box asking for the customer's ID, first name, and last name. After the user enters the information, the function checks that none of the boxes were left empty. If any field is missing, it shows an error message and stops. If all inputs are valid, the function attempts to insert the new customer into the Customers table using an SQL INSERT command. The try-except block handles errors: if the customer ID already exists, it shows a specific database error, and if any other database issue occurs, an exception box is shown. If everything is successful, the data is saved with conn.commit(), and a success pop up message tells the user the customer was added successfully.

```
def show_customers(cursor):
    # Gets all rows from the Customers table and stores them in 'rows'
    cursor.execute("SELECT * FROM Customers")
    rows = cursor.fetchall()

    # If no customers are found, show a message and exit the function
    if not rows:
        eg.msgbox("No customers found.")
        return
    # Creates a formatted text list of all customers for display
    text = "Customer ID\tFirst Name\tLast Name\n" + "="*60 + "\n"
    for customer_id, first_name, last_name in rows:
        text += f"{customer_id}\t{first_name}\t{last_name}\n"

    eg.codebox("Customers", "Customers List", text)



    except sqlite3.Error as e:
        # Catch and display any errors during the retrieval process.
        eg.exceptionbox(msg=f"Failed to retrieve contacts: {e}", title="Database Error")
```

This function retrieves and displays all customers stored in the Customers table. It first executes an SQL select query to get all rows from the table and stores them in the variable rows. If no rows are found, it shows a message informing the user that there are no customers and exits the function. If customers exist, the function creates a formatted text list with column headers for customer ID, first name, and last name, then loops through each row to add the customer information to the list. Finally, it displays the list in a scrollable EasyGUI codebox. A try-except block ensures that any database errors are caught and displayed to the user as an exception message.

```
def add_order(conn, cursor):
    """
    Prompts the user for a new order's details and inserts them into the Customers
table.
    """
    msg = "Enter new order details"
    title = "Add Order"
    fieldNames = ["order_id", "customer_id", "order_date", "total_amount"]

    fieldValues = eg.multenterbox(msg, title, fieldNames)

    if fieldValues is None:
        return

    order_id, customer_id, order_date, total_amount = fieldValues

    # Checks if any input is empty; shows error and stops if so
    if not order_id or not customer_id or not order_date or not total_amount:
        eg.msgbox("All fields are required.", "Input Error")
        return
    # Tries to add a new row to the order table with the provided values
    try:
        cursor.execute("INSERT INTO Orders VALUES (?, ?, ?, ?)",
                       (order_id, customer_id, order_date, total_amount))
        conn.commit()
        eg.msgbox("Order was added successfully!", "Success")

    except sqlite3.IntegrityError:
        eg.msgbox(f"Error: Order ID '{order_id}' already exists.", "Database Error")

    except sqlite3.Error as e:
        eg.exceptionbox(msg=f"Failed to add order: {e}", title="Database Error")
```

This function prompts the user to enter details for a new order and then inserts that information into the Orders table. It shows a pop-up input form asking the user to enter the order ID, customer ID, order date, and total amount. If the user cancels the input or leaves any field blank, a message appears notifying them that all fields are required, and the function exits. If valid data is entered, the function tries to insert a new row into the database with the provided values and commits the changes. A message box informs the user that the order has been added successfully. The try–except block ensures that if the order ID already exists or another database error occurs, the user is shown an appropriate error message.

```
def show_orders(cursor):
    # Gets all rows from the Orders table and stores them in 'rows'
    cursor.execute("SELECT * FROM Orders")
    rows = cursor.fetchall()

    # If no orders are found, show a message and exit the function
    if not rows:
        eg.msgbox("No orders found.")
        return

    # Creates a formatted text list of all orders for display
    text = "Order ID\tCustomer ID\tDate\tAmount\n" + "="*60 + "\n"
    for order_id, customer_id, order_date, total_amount in rows:
        text += f"{order_id}\t{customer_id}\t{order_date}\t{total_amount}\n"

    eg.codebox("Orders", "Orders List", text)
```

The show_orders function retrieves and displays all the orders stored in the database. It first executes a SQL query to select all rows from the Orders table and stores them in the variable rows. If there are no orders in the database, it shows a pop-up message saying “No orders found” and exits the function. Otherwise, it creates a formatted list of all orders, including the order ID, customer ID, order date, and total amount. Finally, it displays this list in a scrollable code box using EasyGUI’s codebox, allowing the user to view all orders neatly in one window.

```
# --- Main Program Logic ---
if __name__ == "__main__":
    # Initialize the database connection and cursor.
    conn, cursor = setup_database()

    # Exit if the database setup failed.
    if not conn:
        exit()

    while True:
        # Use EasyGui's buttonbox to create a main menu for the user.
        # This function returns the text of the button that was clicked.
        choice = eg.buttonbox(
            "What would you like to do?",
            "Main Menu",
            choices=["Add Customer", "Add Order", "Show Customers", "Show Orders",
"Exit"]
        )

        # Handle the user's choice.
        if choice == "Add Customer":
            add_customer(conn, cursor)
        elif choice == "Add Order":
            add_order(conn, cursor)
        elif choice == "Show Customers":
            show_customers(cursor)
        elif choice == "Show Orders":
            show_orders(cursor)
        # The loop breaks if the user clicks 'Exit' or closes the window ('None').
        else:
            break

    # Closes the database connection when the program's main loop ends.
    conn.close()
    eg.msgbox("Goodbye!", "Exiting Program")
```

This section of code runs the main program. It first sets up the database by connecting to it and getting a cursor for executing SQL commands. If the connection fails, the program exits. Then, it enters an infinite loop to create a main menu using EasyGUI’s buttonbox, which presents the user with options: add a customer, add an order, show all customers, show all orders, or exit. Depending on the user’s selection, it calls the corresponding function to perform the chosen action. The loop continues until the user clicks exit or closes the window, at which point the database connection is safely closed, and a final pop-up message appears saying “Goodbye!” to signal the program has ended.