Explaining sections of code

```
# --- 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('Football_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 contacts (
                id INTEGER PRIMARY KEY,
                Name TEXT NOT NULL,
                Position TEXT NOT NULL,
                Player_number INTEGER,
                Goals INTEGER,
                Assists INTEGER
            )
        ''')

        # 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
```

This code defines a function called setup_database() that connects the program to a SQLite database named Football_Database.db. If the database file doesn't already

exist, it automatically creates one. Inside the database, it makes sure there's a table called contacts where player information can be stored — including fields for name, position, player number, goals, and assists. The function then saves these changes and returns the connection and cursor objects, which are used later to run database commands. If something goes wrong while connecting or creating the table, it shows an error message using EasyGui and returns None to indicate that a problem occurred.

```
def add_player(conn, cursor):
    """
    Prompts the user for a new contact's details and inserts them into the database.
    """
    msg = "Enter information about your new player"
    title = "Add Player"
    fieldNames = ["Player_name", "Position", "Player_number", "Goals", "Assists"]

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

    # Use EasyGui's multinterbox to get multiple inputs from the user.
    # The return value is a list of strings, or None if the user clicks 'Cancel'.
    if fieldValues is None:
        return

    # Unpack the list of values into separate variables.
    Player_name, Position, Player_number, Goals, Assists = fieldValues

    # Validate that both fields have been filled in.
    if not Player_name or not Position or not Player_number or not Goals or not
Assists:
        eg.msgbox("All of the boxes need to be filled!", "Input Error")
        return

    try:
        # Execute an INSERT SQL command using placeholders (?) to prevent SQL
injection.
        # This is the safest way to insert user-provided data.
        cursor.execute("INSERT INTO Football_Database (Name, Position,
Player_number, Goals, Assists) VALUES (?, ?, ?, ?, ?)",
        (Player_name, Position, Player_number, Goals, Assists))

        # Commit the changes to the database to make the insertion permanent.
        conn.commit()
```

```
        # Show a success message to the user.
        eg.msgbox(f"Player '{Player_name}' added successfully!", "Success")

    except sqlite3.IntegrityError:
        # Catch a specific error if the email is not unique (due to the UNIQUE
constraint).
        eg.msgbox(f"Error: The player '{Player_name}' already exists.", "Database
Error")
    except sqlite3.Error as e:
        # Catch any other database errors and display them.
        eg.exceptionbox(msg=f"Failed to add contact: {e}", title="Database Error")
```

This function, add_player(), allows the user to add a new player's details to the database. It opens a pop-up form using EasyGui, where the user enters the player's name, position, number, goals, and assists. If the user cancels the form or leaves any boxes empty, the function stops and shows an error message. Otherwise, it takes the entered information and safely inserts it into the Football_Database table using an SQL INSERT command. After saving the new data permanently with conn.commit(), it displays a success message confirming the player was added. If something goes wrong — like a duplicate entry or another database issue — the function shows an error message explaining what happened.

```
def show_Players(cursor):
    """
    Retrieves all Player from the database and displays them in a formatted message
box.
    """
    try:
        # Execute a SELECT query to get the Player name and Details of all contacts.
        cursor.execute("SELECT Name, Position, Player_number, Goals, Assists FROM
Football_Database")

        # Fetch all the results from the query as a list of tuples.
        rows = cursor.fetchall()

        if not rows:
            # If the list is empty, there are no contacts.
            eg.msgbox("No players found in the database.", "Player List")
            return

        # Prepare a header for the display text.
```

```
        Player_list = "Name\t\tPosition\t\tNumber\t\tGoals\t\tAssists\n"
        Player_list += "=" * 80 + "\n"

        # Loop through the list of tuples and format each contact into a string.
        for name, position, player_number, goals, assists in rows:
            Player_list +=
f"{name}\t\t{position}\t\t{player_number}\t\t{goals}\t\t{assists}\n"

        # Use EasyGui's textbox to show the formatted list. This widget is scrollable.
        eg.codebox("All Players", "Player List", Player_list)

    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, show_Players(), displays all the players stored in the database in a neat, scrollable pop-up window. It first runs an SQL SELECT command to collect each player's name, position, number, goals, and assists from the Football_Database table. The results are stored in a list called rows. If the list is empty, meaning no players exist, it shows a message saying so. Otherwise, it formats all the player details into a clean, table-like text layout and shows them using EasyGui's codebox, which allows scrolling through the list. If there's a problem while retrieving the data, it catches the database error and displays an error message explaining what went wrong.

```
# --- 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 Player", "Show All Players", "Exit"]
        )
```

```
    # Handle the user's choice.
    if choice == "Add Player":
        add_player(conn, cursor)
    elif choice == "Show All Players":
        show_Players(cursor)
    elif choice == "Exit" or choice is None:
        # The loop breaks if the user clicks 'Exit' or closes the window ('None').
        break

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

This is the main part of the program that runs when the script is executed. It first sets up the database by calling setup_database() and gets the connection and cursor objects needed to interact with the database. If the database setup fails, the program exits. Otherwise, it enters a loop that repeatedly shows a pop-up menu with three options: “Add Player,” “Show All Players,” and “Exit.” Depending on what the user clicks, it either calls add_player() to add a new player, show_Players() to display all players, or breaks the loop to exit the program. When the user chooses to exit, it safely closes the database connection and shows a “Goodbye!” message before ending.