

Data Processing and Database Project

By

Juan Moctezuma

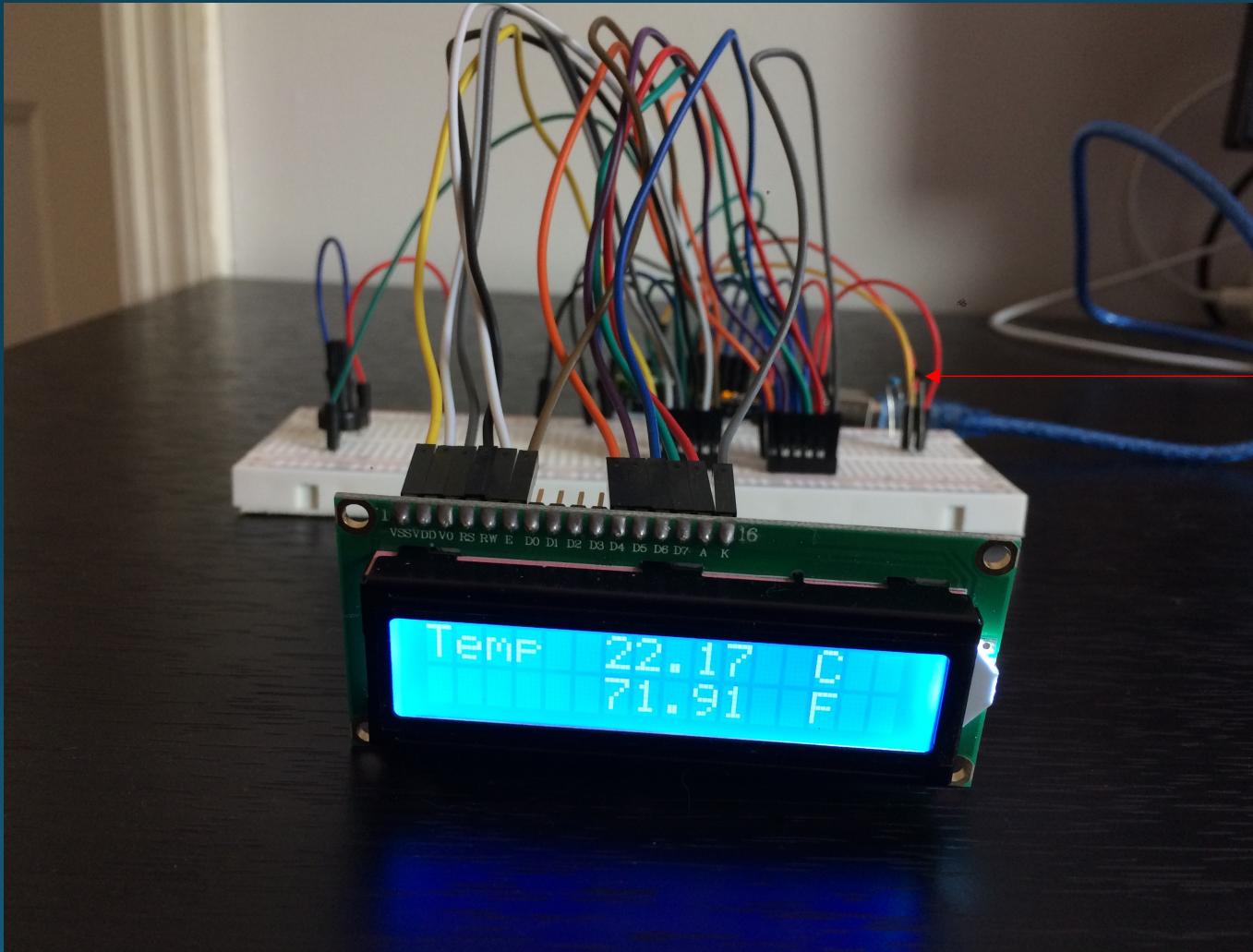
What is this project about?

- Demonstrating how raw data (room temperature) can be obtained from a source; in this case a device.
- Preparing and modifying the information obtained directly from the source throughout a series of steps.
- Making use of the available technological tools required to develop a final product (an Excel spreadsheet with results nicely organized and displayed) while maintaining high quality data.
- Creating a database (hence a SQL table) for storage or record keeping purposes.

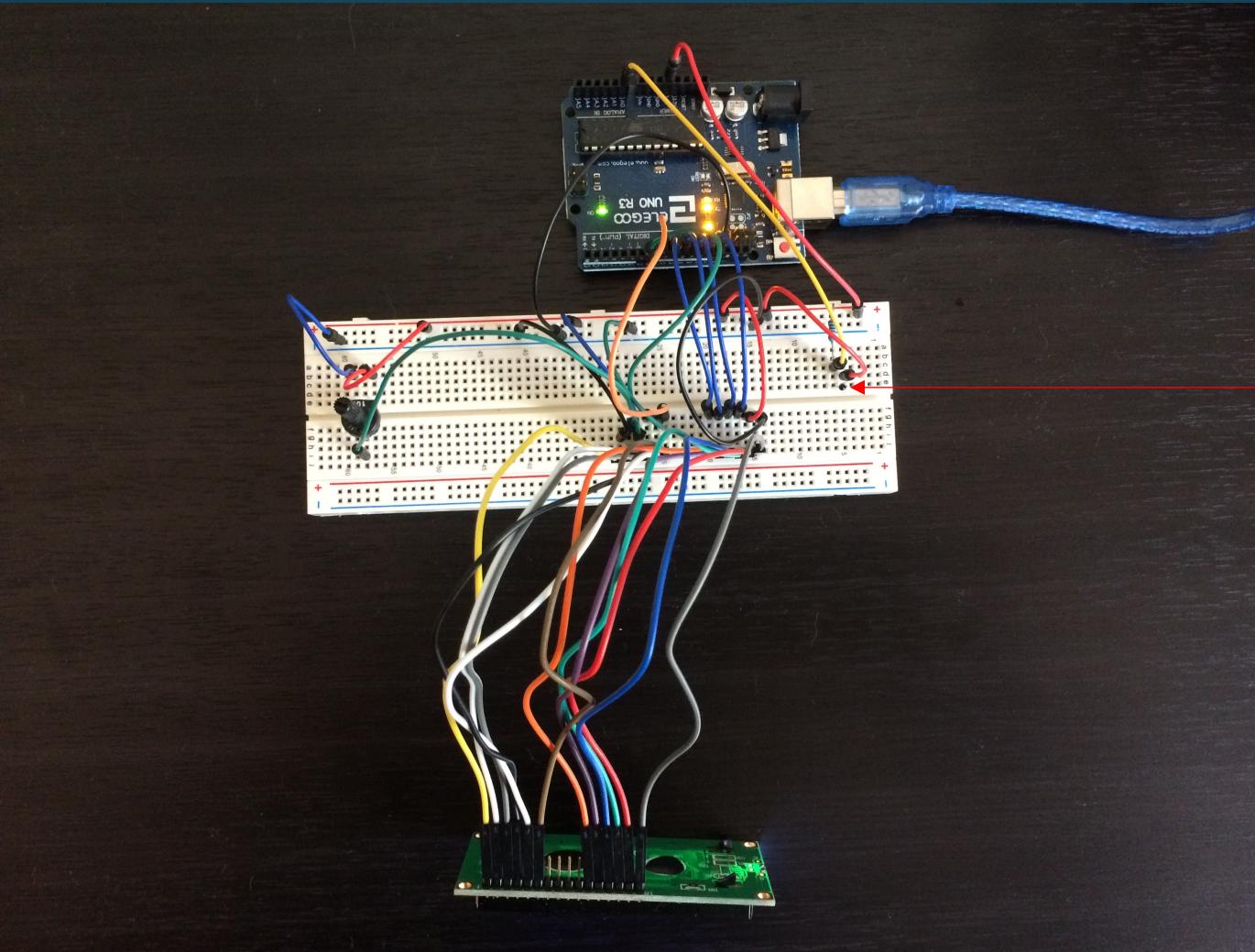
Where is the raw data coming from?

- The source producing data or information is a temperature sensor attached to an *Arduino UNO R3* circuit board and other essential electrical components. Basically a digital thermometer. Please note that this project will not go into details related with Electrical Engineering or the principles of electricity.
- The 3 following slides will show images of the actual thermometer, temperature sensor and the connection schematic (instructions of how the device needs to be physically connected).

Digital Thermometer – Front View



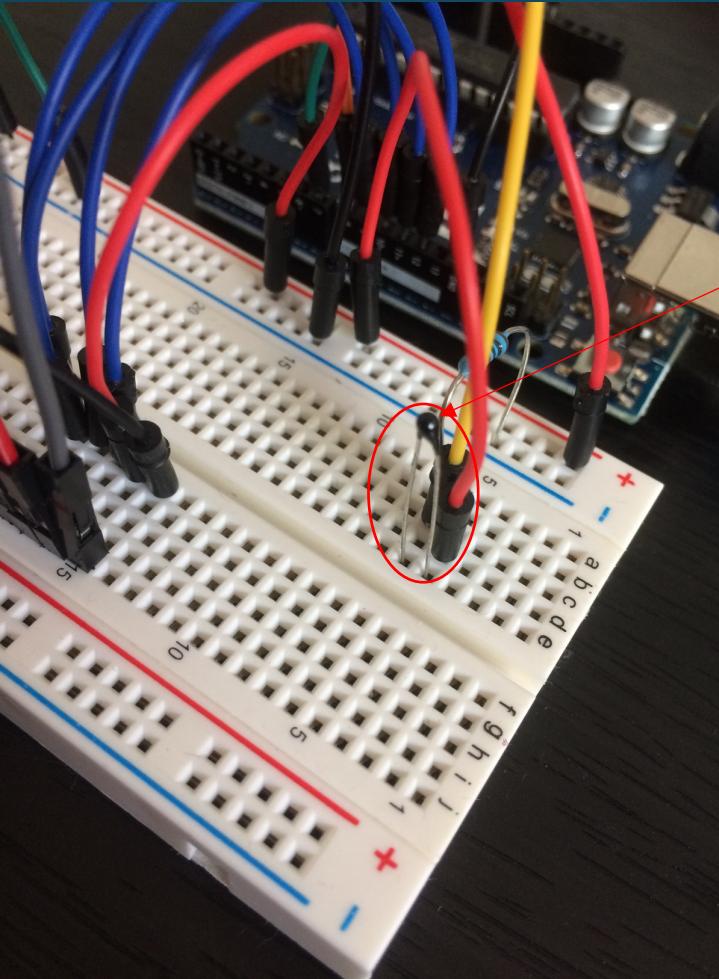
Digital Thermometer – Top View



Temperature Sensor

Temperature Sensor

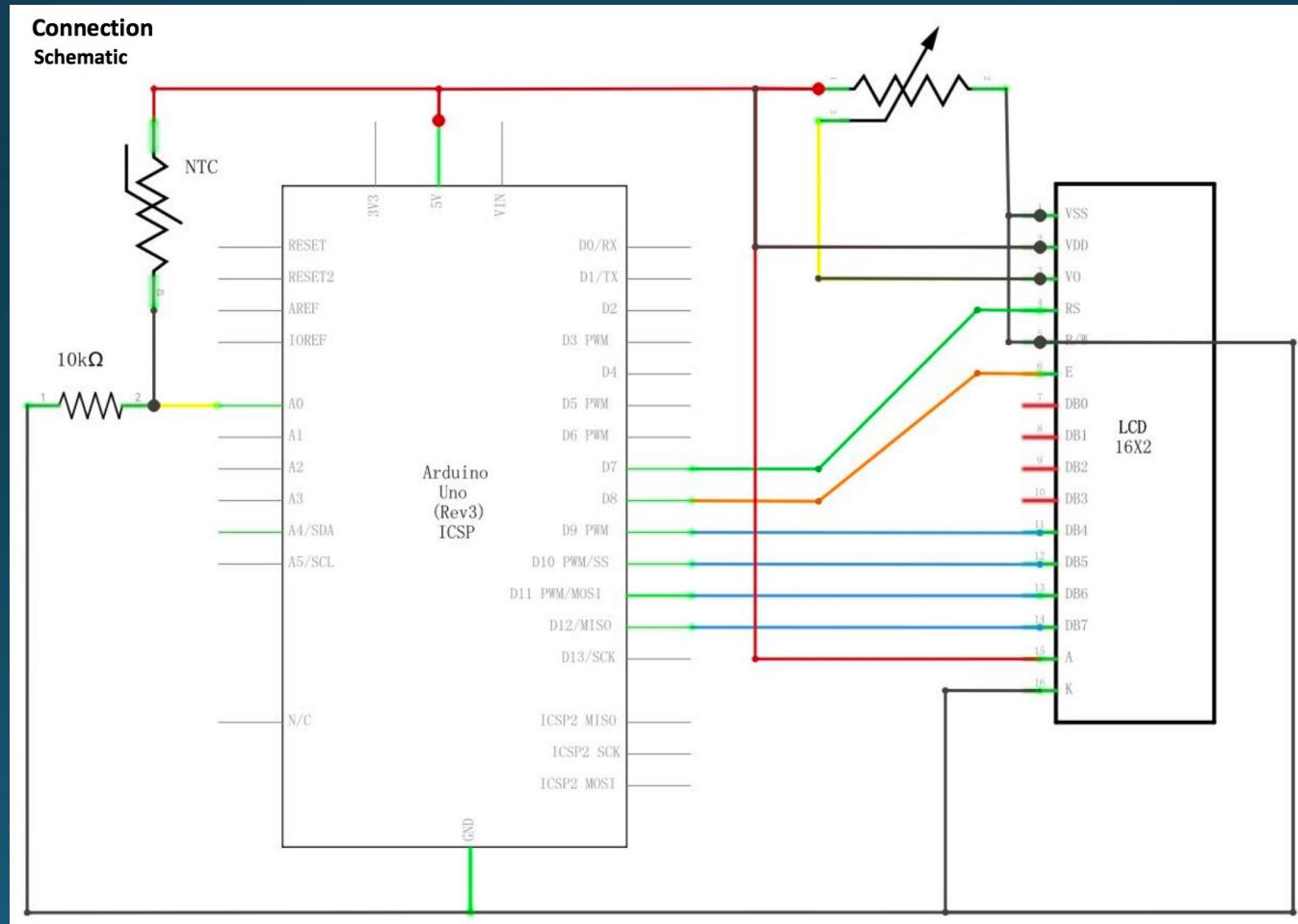
- Readings are measured by the small black structure within the red oval.



Note: If this black component is touched directly, readings will reflect body temperature instead of room temperature.

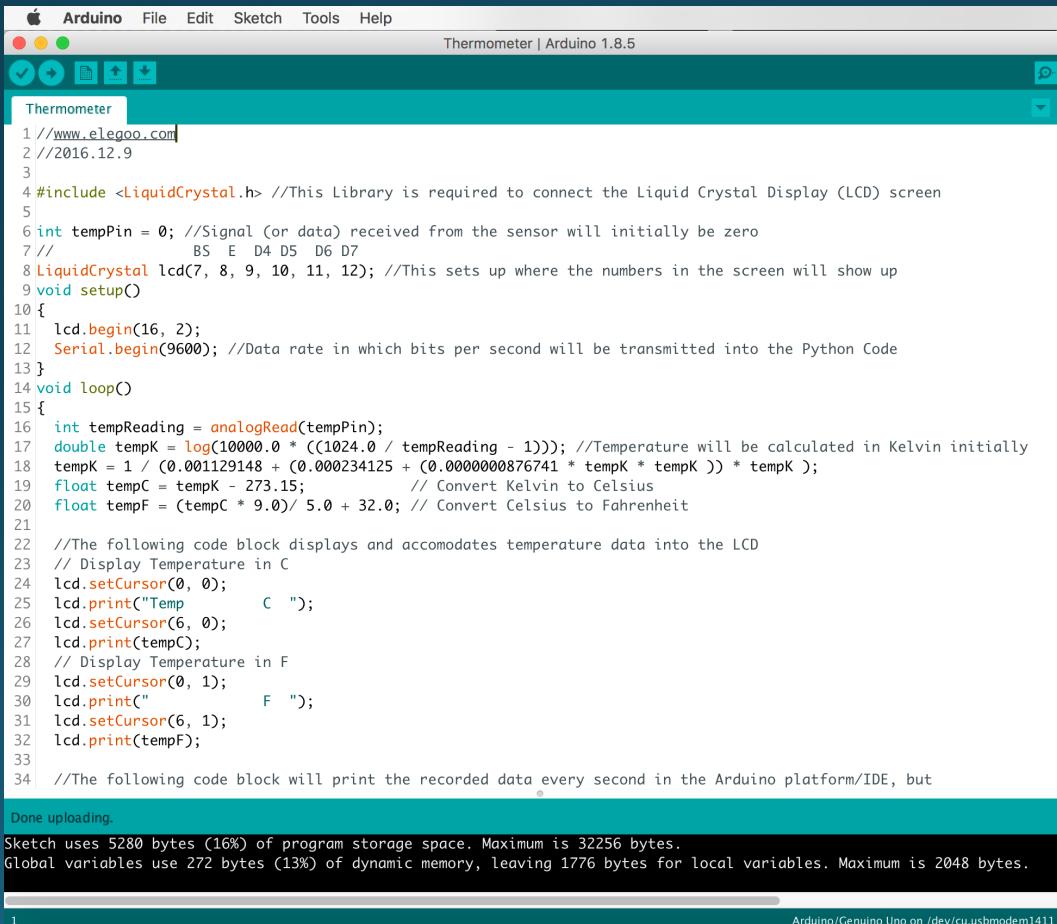
Electric Diagram or Schematic

- Note: The following image was retrieved from *Elegoo Super Starter Kit for UNO*. I did not create or modified the picture provided in this slide.



How do we make our thermometer work?

- After the device is wired up and connected to a computer via USB, we need to open Arduino's open-source Integrated Development Environment (IDE) and run the code written in C/C++.



The screenshot shows the Arduino IDE interface with the title bar "Thermometer | Arduino 1.8.5". The main window displays the following C/C++ code for a thermometer project:

```
1 //www.elegoo.com
2 //2016.12.9
3
4 #include <LiquidCrystal.h> //This Library is required to connect the Liquid Crystal Display (LCD) screen
5
6 int tempPin = 0; //Signal (or data) received from the sensor will initially be zero
7 //           BS E D4 D5 D6 D7
8 LiquidCrystal lcd(7, 8, 9, 10, 11, 12); //This sets up where the numbers in the screen will show up
9 void setup()
10 {
11   lcd.begin(16, 2);
12   Serial.begin(9600); //Data rate in which bits per second will be transmitted into the Python Code
13 }
14 void loop()
15 {
16   int tempReading = analogRead(tempPin);
17   double tempK = log(10000.0 * ((1024.0 / tempReading - 1))); //Temperature will be calculated in Kelvin initially
18   tempK = 1 / (0.001129148 + (0.000234125 + (0.000000876741 * tempK * tempK)) * tempK );
19   float tempC = tempK - 273.15; // Convert Kelvin to Celsius
20   float tempF = (tempC * 9.0)/ 5.0 + 32.0; // Convert Celsius to Fahrenheit
21
22 //The following code block displays and accomodates temperature data into the LCD
23 // Display Temperature in C
24 lcd.setCursor(0, 0);
25 lcd.print("Temp      C  ");
26 lcd.setCursor(6, 0);
27 lcd.print(tempC);
28 // Display Temperature in F
29 lcd.setCursor(0, 1);
30 lcd.print("      F  ");
31 lcd.setCursor(6, 1);
32 lcd.print(tempF);
33
34 //The following code block will print the recorded data every second in the Arduino platform/IDE, but
```

The status bar at the bottom shows "Done uploading." and provides memory usage information: "Sketch uses 5280 bytes (16%) of program storage space. Maximum is 32256 bytes. Global variables use 272 bytes (13%) of dynamic memory, leaving 1776 bytes for local variables. Maximum is 2048 bytes."

What role does Python play?

- We need Python 3.6 in order to establish a direct connection with the digital thermometer and obtain its data or measurements. This needs to occur as the code from the other device is running at the same time as well.
- Once these are connected and PyCharm (Python's platform or IDE) starts receiving live/real time data, each record (temperature in Celsius and Fahrenheit) will get joined along with a timestamp and then separated by a comma; timestamps also get split into date and time, and separated by a comma.
 - For instance, if room temperature was 70° at some point in time, then the record gets compiled as "21.47,70.64,05/28/2019,17:59:05".



What will happen to the compiled room temperature records?

- The Python script is also programmed to gather each record into a text file conveniently named *Arduino_Temperature_Raw_Data.txt*, and then get that file exported into my personal computer's Documents folder.
- This text file containing the raw data is the Python code's 'output', which will continue to modified (with Microsoft Excel – Macro/VBA) after recording the desired amount of data.
- For the sake of simplicity only 2-day worth of data was obtained by leaving the python script along with the thermometer running for several hours during two consecutive days.

How does the Python script and IDE look like?

Each record gets displayed every second in the same format as in the example provided in slide 9. As long as the thermometer is running, data won't stop compiling.

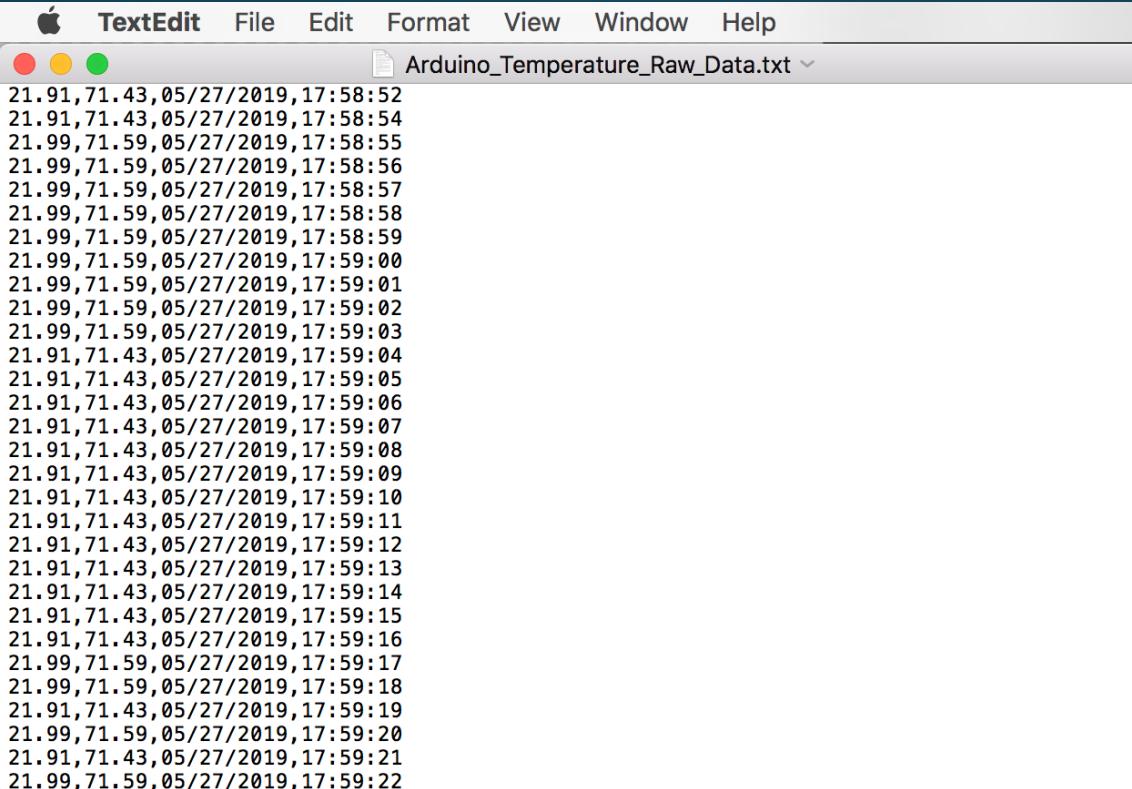
The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "Thermometer_Data_Converter". It contains a virtual environment folder ("venv") which includes "bin", "include", and "lib" subfolders, along with files "Arduino_Serial_Monitor_Txt_Converter.py", "Arduino_Temperature_Raw_Data.txt", "output.txt", and "pyenv.cfg".
- Code Editor:** The main editor window displays the Python script "Arduino_Serial_Monitor_Txt_Converter.py". The code reads serial data from an Arduino connected via USB, decodes it, and writes the timestamped temperature data to a text file ("Arduino_Temperature_Raw_Data.txt"). It handles errors and keyboard interrupts.
- Run Tab:** The bottom tab bar shows the run configuration: "Arduino_Serial_Monitor_Txt_Converter". A red circle highlights the output terminal window.
- Output Terminal:** The terminal window shows the recorded data:

```
22.43,72.38,05/29/2019,12:28:42
22.43,72.38,05/29/2019,12:28:43
22.43,72.38,05/29/2019,12:28:44
22.35,72.22,05/29/2019,12:28:45
22.35,72.22,05/29/2019,12:28:46
22.35,72.22,05/29/2019,12:28:47
22.35,72.22,05/29/2019,12:28:48
22.35,72.22,05/29/2019,12:28:49
22.35,72.22,05/29/2019,12:28:50
22.43,72.38,05/29/2019,12:28:51
22.43,72.38,05/29/2019,12:28:52
```

How does the output look?

- As mentioned previously, the output containing the raw data is simply a list within *Arduino_Temperature_Raw_Data.txt* containing every record.



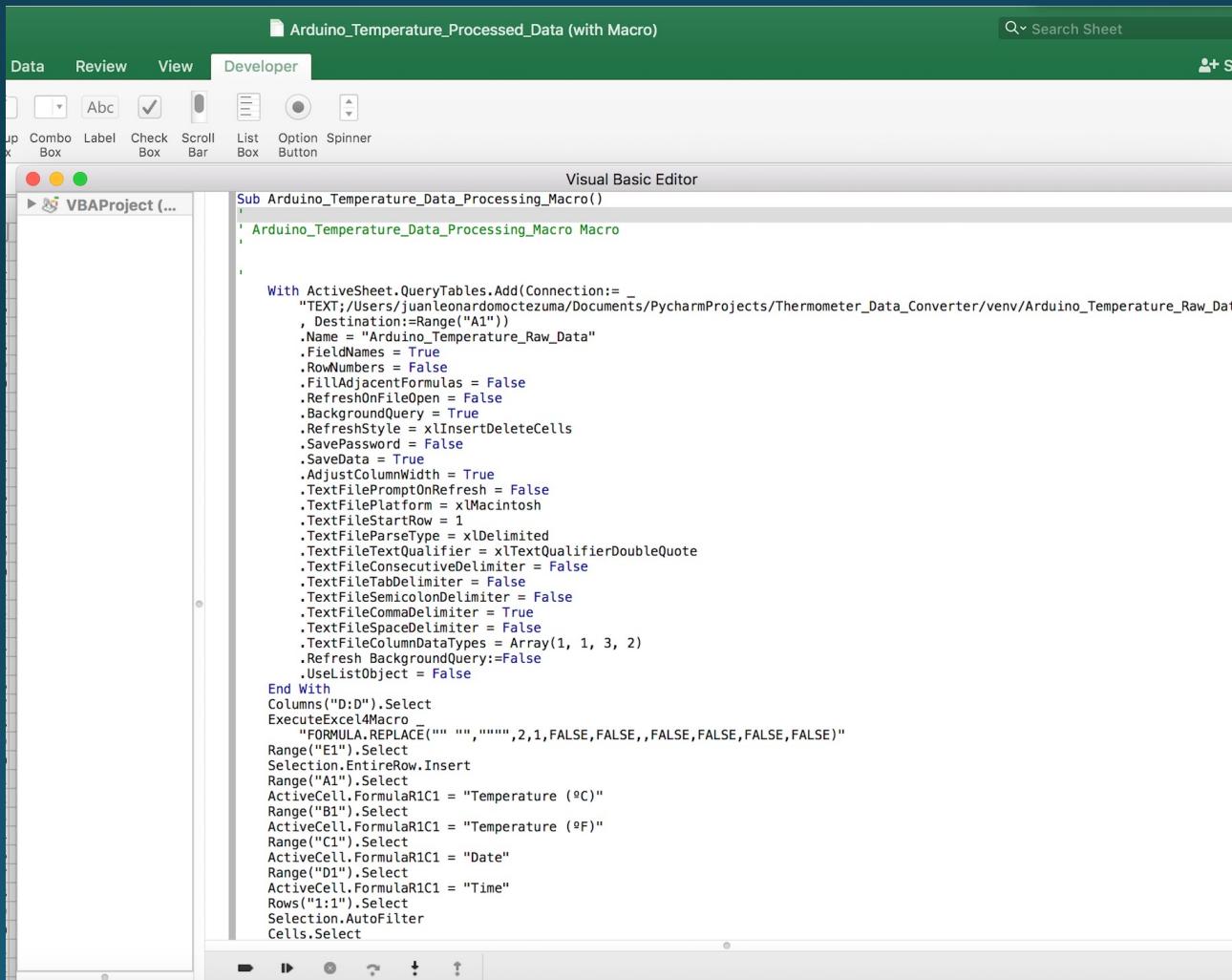
The screenshot shows a Mac OS X TextEdit window with the title bar "TextEdit" and the file name "Arduino_Temperature_Raw_Data.txt". The window contains a single column of text representing raw temperature data. Each line consists of two values separated by a comma: a float value (either 21.91 or 21.99) and a timestamp (e.g., 71.43, 05/27/2019, 17:58:52). There are 30 such records listed.

| Value | Date/Time |
|--------------|----------------------|
| 21.91, 71.43 | 05/27/2019, 17:58:52 |
| 21.91, 71.43 | 05/27/2019, 17:58:54 |
| 21.99, 71.59 | 05/27/2019, 17:58:55 |
| 21.99, 71.59 | 05/27/2019, 17:58:56 |
| 21.99, 71.59 | 05/27/2019, 17:58:57 |
| 21.99, 71.59 | 05/27/2019, 17:58:58 |
| 21.99, 71.59 | 05/27/2019, 17:58:59 |
| 21.99, 71.59 | 05/27/2019, 17:59:00 |
| 21.99, 71.59 | 05/27/2019, 17:59:01 |
| 21.99, 71.59 | 05/27/2019, 17:59:02 |
| 21.99, 71.59 | 05/27/2019, 17:59:03 |
| 21.91, 71.43 | 05/27/2019, 17:59:04 |
| 21.91, 71.43 | 05/27/2019, 17:59:05 |
| 21.91, 71.43 | 05/27/2019, 17:59:06 |
| 21.91, 71.43 | 05/27/2019, 17:59:07 |
| 21.91, 71.43 | 05/27/2019, 17:59:08 |
| 21.91, 71.43 | 05/27/2019, 17:59:09 |
| 21.91, 71.43 | 05/27/2019, 17:59:10 |
| 21.91, 71.43 | 05/27/2019, 17:59:11 |
| 21.91, 71.43 | 05/27/2019, 17:59:12 |
| 21.91, 71.43 | 05/27/2019, 17:59:13 |
| 21.91, 71.43 | 05/27/2019, 17:59:14 |
| 21.91, 71.43 | 05/27/2019, 17:59:15 |
| 21.91, 71.43 | 05/27/2019, 17:59:16 |
| 21.99, 71.59 | 05/27/2019, 17:59:17 |
| 21.99, 71.59 | 05/27/2019, 17:59:18 |
| 21.91, 71.43 | 05/27/2019, 17:59:19 |
| 21.99, 71.59 | 05/27/2019, 17:59:20 |
| 21.91, 71.43 | 05/27/2019, 17:59:21 |
| 21.99, 71.59 | 05/27/2019, 17:59:22 |

What will the Excel Macro do to the output text file?

- Once the raw info was imported in Microsoft Excel as a text file, a Macro recorded every step that was performed in order to obtain the processed (completed) data.
- The steps were as followed:
 - Created columns based on delimiters (commas).
 - Replaced every blank space created after Time column (Column D) was imported. Note: I wasn't able to figure out why my Python script was creating an unwanted space in every record, therefore this step was required to be done in Excel in order to ensure quality and avoid formatting-related errors.
 - For instance: "17:58:52 " has an extra place to the right of number two, which was replaced with no blank spaces at all, such as "17:58:52".
 - Inserted four headers in the first row and assigned them filters. Headers consist of Temperature (°C), Temperature (°F), Date and Time.

How does the Macro look like?

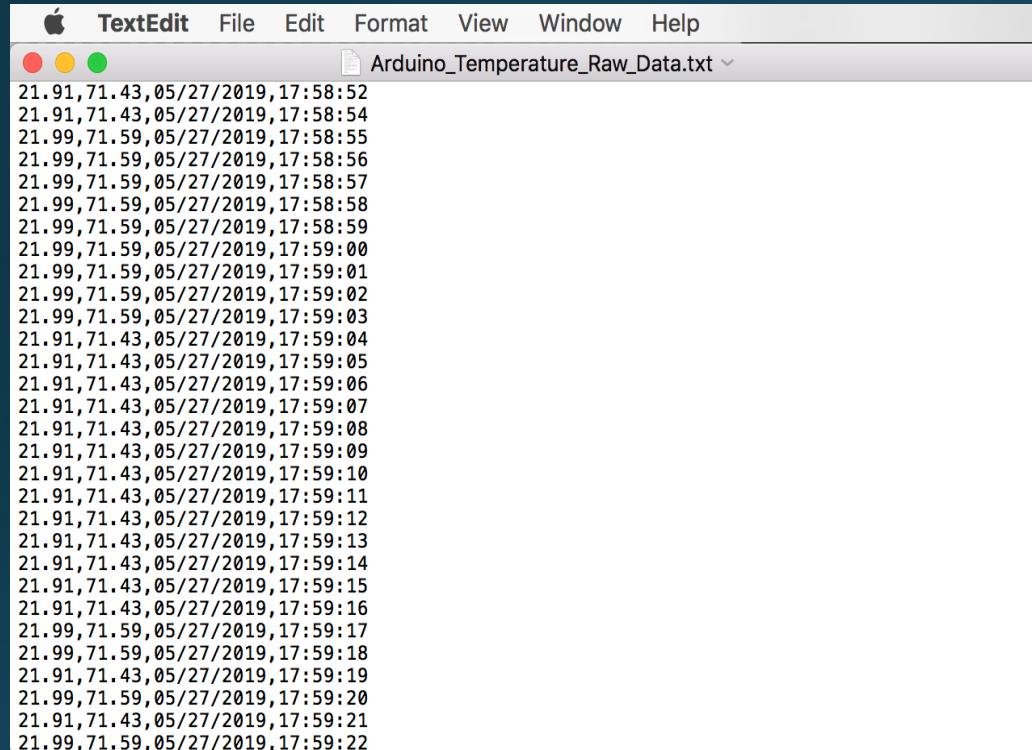


The screenshot shows the Microsoft Excel Visual Basic Editor (VBE) interface. The title bar reads "Arduino_Temperature_Processed_Data (with Macro)". The menu bar includes "Data", "Review", "View", and "Developer". The "Developer" tab is selected, showing various tool icons: Up Box, Combo Box, Label, Check Box, Scroll Bar, List Box, Option Button, and Spinner. Below the menu is the "Visual Basic Editor" window. The code listed is:

```
Sub Arduino_Temperature_Data_Processing_Macro()
    ' Arduino_Temperature_Data_Processing Macro

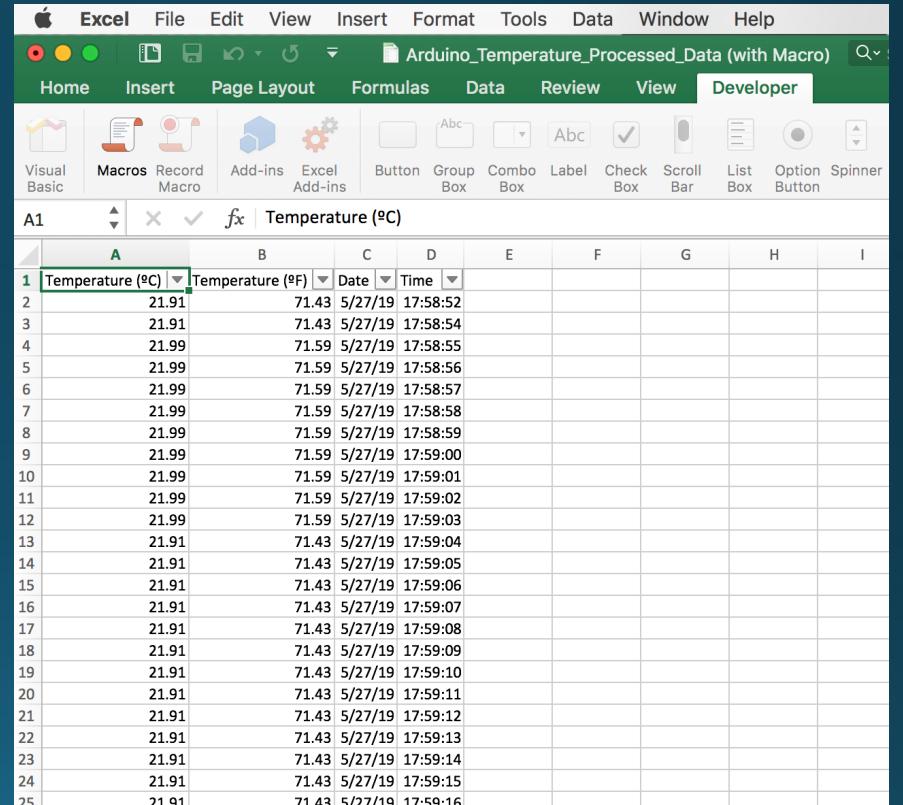
    With ActiveSheet.QueryTables.Add(Connection:=_
        "TEXT; /Users/juanleonardomocetzuma/Documents/PycharmProjects/Termometer_Data_Converter/venv/Arduino_Temperature_Raw_Data"
        , Destination:=Range("A1"))
        .Name = "Arduino_Temperature_Raw_Data"
        .FieldNames = True
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .RefreshOnFileOpen = False
        .BackgroundQuery = True
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = True
        .TextFilePromptOnRefresh = False
        .TextFilePlatform = xlMacintosh
        .TextFileStartRow = 1
        .TextFileParseType = xlDelimited
        .TextFileTextQualifier = xlTextQualifierDoubleQuote
        .TextFileConsecutiveDelimiter = False
        .TextFileTabDelimeter = False
        .TextFileSemicolonDelimeter = False
        .TextFileCommaDelimeter = True
        .TextFileSpaceDelimeter = False
        .TextFileColumnDataTypes = Array(1, 1, 3, 2)
        .Refresh BackgroundQuery:=False
        .UseListObject = False
    End With
    Columns("D:D").Select
    ExecuteExcel4Macro
    "FORMULA.REPLACE("" "" , "" , 2, 1, FALSE, FALSE, , FALSE, FALSE, FALSE, FALSE)"
    Range("E1").Select
    Selection.EntireRow.Insert
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Temperature (°C)"
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Temperature (°F)"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "Date"
    Range("D1").Select
    ActiveCell.FormulaR1C1 = "Time"
    Rows("1:1").Select
    Selection.AutoFilter
    Cells.Select
```

Temperature Records: Before and After



Raw data: Python script's output after compiling Measurements. Please note that each data point is separated by commas and each row represents an element of a list.

```
21.91,71.43,05/27/2019,17:58:52
21.91,71.43,05/27/2019,17:58:54
21.99,71.59,05/27/2019,17:58:55
21.99,71.59,05/27/2019,17:58:56
21.99,71.59,05/27/2019,17:58:57
21.99,71.59,05/27/2019,17:58:58
21.99,71.59,05/27/2019,17:58:59
21.99,71.59,05/27/2019,17:59:00
21.99,71.59,05/27/2019,17:59:01
21.99,71.59,05/27/2019,17:59:02
21.99,71.59,05/27/2019,17:59:03
21.91,71.43,05/27/2019,17:59:04
21.91,71.43,05/27/2019,17:59:05
21.91,71.43,05/27/2019,17:59:06
21.91,71.43,05/27/2019,17:59:07
21.91,71.43,05/27/2019,17:59:08
21.91,71.43,05/27/2019,17:59:09
21.91,71.43,05/27/2019,17:59:10
21.91,71.43,05/27/2019,17:59:11
21.91,71.43,05/27/2019,17:59:12
21.91,71.43,05/27/2019,17:59:13
21.91,71.43,05/27/2019,17:59:14
21.91,71.43,05/27/2019,17:59:15
21.91,71.43,05/27/2019,17:59:16
21.99,71.59,05/27/2019,17:59:17
21.99,71.59,05/27/2019,17:59:18
21.91,71.43,05/27/2019,17:59:19
21.99,71.59,05/27/2019,17:59:20
21.91,71.43,05/27/2019,17:59:21
21.99,71.59,05/27/2019,17:59:22
```



Processed data: Resulting spreadsheet after running macro. This would be the FINAL output.

| | A | B | C | D | E | F | G | H | I |
|----|------------------|------------------|---------|----------|---|---|---|---|---|
| 1 | Temperature (°C) | Temperature (°F) | Date | Time | | | | | |
| 2 | 21.91 | 71.43 | 5/27/19 | 17:58:52 | | | | | |
| 3 | 21.91 | 71.43 | 5/27/19 | 17:58:54 | | | | | |
| 4 | 21.99 | 71.59 | 5/27/19 | 17:58:55 | | | | | |
| 5 | 21.99 | 71.59 | 5/27/19 | 17:58:56 | | | | | |
| 6 | 21.99 | 71.59 | 5/27/19 | 17:58:57 | | | | | |
| 7 | 21.99 | 71.59 | 5/27/19 | 17:58:58 | | | | | |
| 8 | 21.99 | 71.59 | 5/27/19 | 17:58:59 | | | | | |
| 9 | 21.99 | 71.59 | 5/27/19 | 17:59:00 | | | | | |
| 10 | 21.99 | 71.59 | 5/27/19 | 17:59:01 | | | | | |
| 11 | 21.99 | 71.59 | 5/27/19 | 17:59:02 | | | | | |
| 12 | 21.99 | 71.59 | 5/27/19 | 17:59:03 | | | | | |
| 13 | 21.91 | 71.43 | 5/27/19 | 17:59:04 | | | | | |
| 14 | 21.91 | 71.43 | 5/27/19 | 17:59:05 | | | | | |
| 15 | 21.91 | 71.43 | 5/27/19 | 17:59:06 | | | | | |
| 16 | 21.91 | 71.43 | 5/27/19 | 17:59:07 | | | | | |
| 17 | 21.91 | 71.43 | 5/27/19 | 17:59:08 | | | | | |
| 18 | 21.91 | 71.43 | 5/27/19 | 17:59:09 | | | | | |
| 19 | 21.91 | 71.43 | 5/27/19 | 17:59:10 | | | | | |
| 20 | 21.91 | 71.43 | 5/27/19 | 17:59:11 | | | | | |
| 21 | 21.91 | 71.43 | 5/27/19 | 17:59:12 | | | | | |
| 22 | 21.91 | 71.43 | 5/27/19 | 17:59:13 | | | | | |
| 23 | 21.91 | 71.43 | 5/27/19 | 17:59:14 | | | | | |
| 24 | 21.91 | 71.43 | 5/27/19 | 17:59:15 | | | | | |
| 25 | 21.91 | 71.43 | 5/27/19 | 17:59:16 | | | | | |

Raw data: Python script's output after compiling Measurements. Please note that each data point is separated by commas and each row represents an element of a list.

Processed data: Resulting spreadsheet after running macro. This would be the FINAL output.

What is the purpose of the Macro?

- To automate some of the steps regarding data manipulation.
- Simulate and redo every step performed manually to a dataset.
Please note that data must have the same structure otherwise the macro will produce erroneous results.
- Save time by avoiding repetitive manual work in Microsoft Excel which most of the time is unnecessary.
- Re-use the same recently created Macro or VBA code in case the data needed an update and/or data points were to be appended to the raw data text file.

What happens to the final output?

- The Excel file now needs to get saved as a CSV file in order to get loaded into a table (a subset of a database).
- Data gets stored for record keeping purposes. Although this step might not affect the quality of our data directly, we still need to have the appropriate formatting for every column. Not doing might cause pgAdmin3 (PostgreSQL) to reject the CSV file depending on the conditions in which that table was created.



Postgre*SQL*

Loading Data into a Table

- As mentioned on previous slide (17), in most cases tables require each data set to meet certain conditions.
- In this project, the table was intentionally set to only accept CVS containing the following:
 - Temperature data (decimal numbers with two places).
 - Date with a valid format (there are several acceptable formats).
 - Time with a valid format (there are several acceptable formats).
 - None of the records can have missing information (blank cells within the spreadsheet).

SQL Table

- The image below demonstrates how the RECORDED_ROOM_TEMPERATURE table was created with SQL syntax and through pgAdmin3:

The screenshot shows the pgAdmin3 interface. The title bar reads "Query - Arduino_Temperature_Database on postgres@localhost:5432 *". The main window displays the SQL Editor tab with the following code:

```
CREATE TABLE RECORDED_ROOM_TEMPERATURE (
    TEMPERATURE_CELSIUS DOUBLE PRECISION NOT NULL,
    TEMPERATURE_FAHRENHEIT DOUBLE PRECISION NOT NULL,
    TEMPERATURE_DATE DATE NOT NULL,
    TEMPERATURE_TIME TIME UNIQUE NOT NULL
);
```

The code is highlighted in blue, indicating it is SQL syntax. Below the editor, there is an "Output pane" with tabs for "Data Output", "Explain", "Messages", and "History". The "Messages" tab is selected, showing the message: "Query returned successfully with no result in 12 msec."

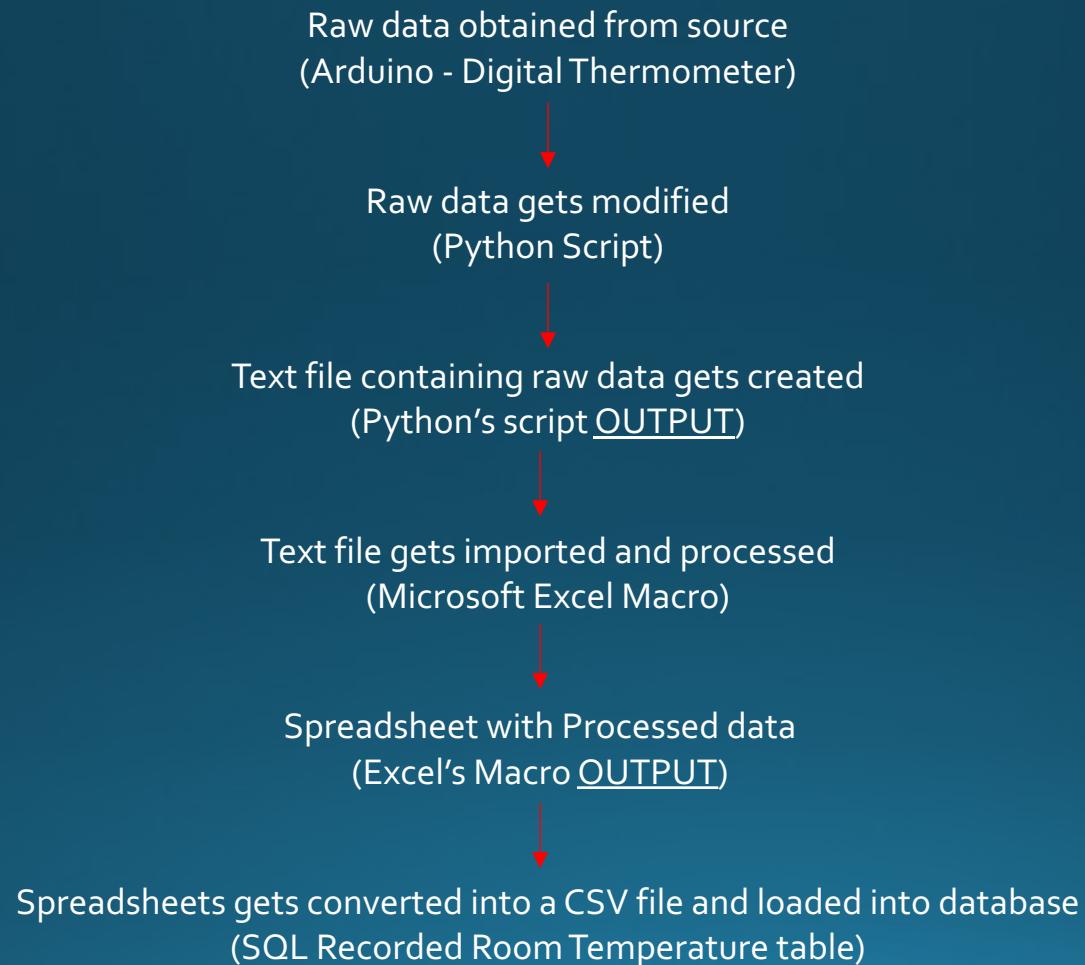
SQL Table (cont.)

- The following picture displays how our data looks like after writing a simple query selecting all data, after being stored into the previously mentioned table:

The screenshot shows the pgAdmin3 interface on a Mac OS X system. The title bar reads "Query - Arduino_Temperature_Database on postgres@localhost:5432". The main window has two panes: "SQL Editor" and "Graphical Query Builder", with "SQL Editor" selected. In the SQL Editor pane, the query "SELECT * FROM RECORDED_ROOM_TEMPERATURE;" is entered. The Output pane below shows the results of the query:

| | temperature_celsius double precision | temperature_fahrenheit double precision | temperature_date date | temperature_time time without time zone |
|----|---|--|--------------------------|--|
| 1 | 21.91 | 71.43 | 2019-05-27 | 17:58:52 |
| 2 | 21.91 | 71.43 | 2019-05-27 | 17:58:54 |
| 3 | 21.99 | 71.59 | 2019-05-27 | 17:58:55 |
| 4 | 21.99 | 71.59 | 2019-05-27 | 17:58:56 |
| 5 | 21.99 | 71.59 | 2019-05-27 | 17:58:57 |
| 6 | 21.99 | 71.59 | 2019-05-27 | 17:58:58 |
| 7 | 21.99 | 71.59 | 2019-05-27 | 17:58:59 |
| 8 | 21.99 | 71.59 | 2019-05-27 | 17:59:00 |
| 9 | 21.99 | 71.59 | 2019-05-27 | 17:59:01 |
| 10 | 21.99 | 71.59 | 2019-05-27 | 17:59:02 |
| 11 | 21.99 | 71.59 | 2019-05-27 | 17:59:03 |
| 12 | 21.91 | 71.43 | 2019-05-27 | 17:59:04 |
| 13 | 21.91 | 71.43 | 2019-05-27 | 17:59:05 |

Summary Diagram



Thank You for Watching!