Data Processing and Database Project

Juan Moctezuma

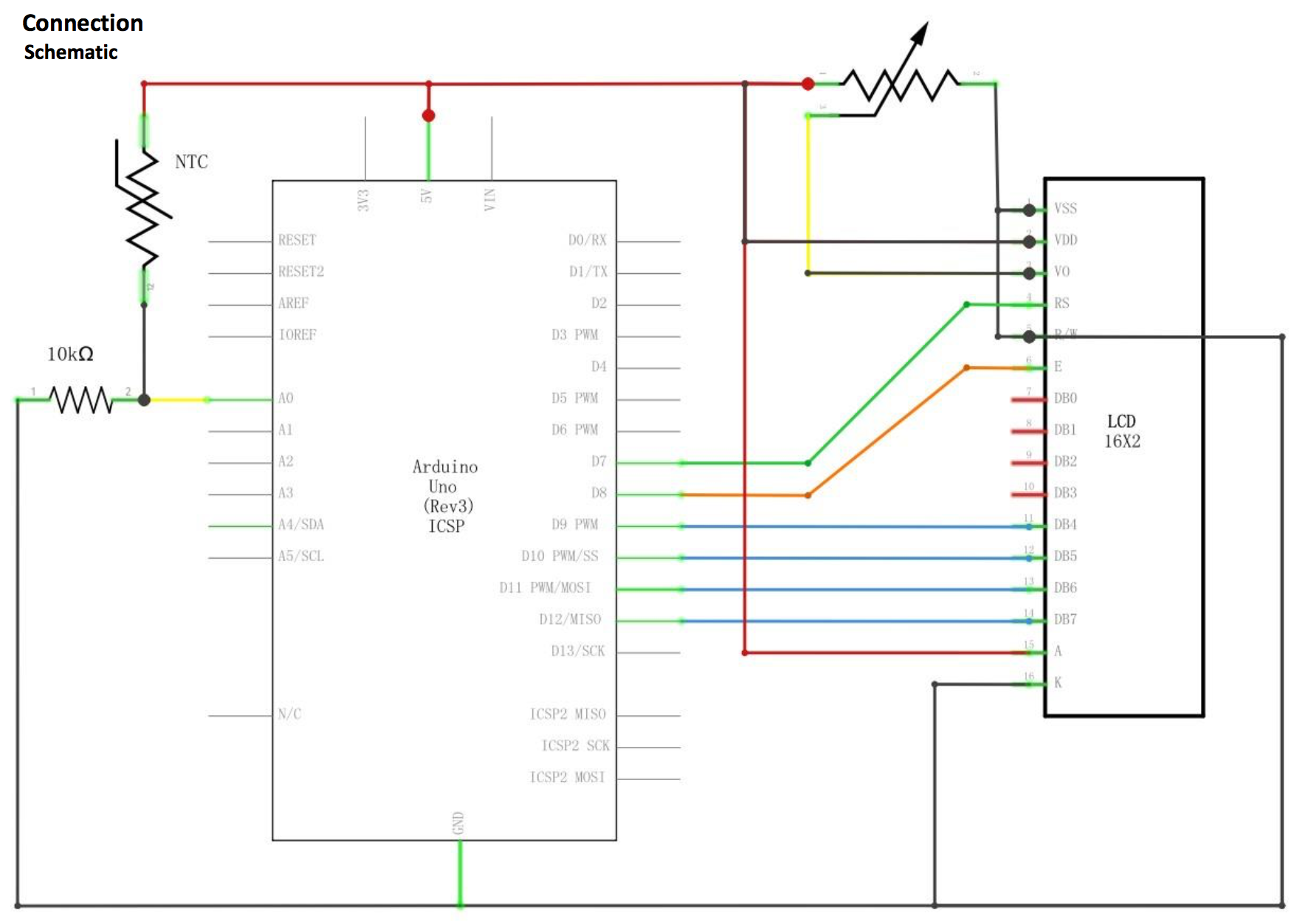
Like in most engineering processes, did you know that most of the time data needs to be prepared and/or modified prior to be released as a final product or to be presented to the user? The purpose of this paper is to describe the steps required to obtain raw data from a source and convert the information into quality, presentable and understandable data available for Microsoft Excel and eventually to get stored into a database. In addition, to get information loaded into a database created with SQL, data requires to be formatted appropriately to assure that it gets stored with no errors.

During this project, a digital thermometer was compiled by wiring up an *Arduino UNO R3* circuit board, a temperature sensor and a few electrical components with the purpose of recording room temperature data during several hours between two consecutive days. Please note that there was no more than 2 day-worth of data since the goal of this project is to demonstrate how quality data was processed and stored, not to prove weather patterns. In addition, other tools and steps were essential to achieve the completion of this process. The following bullets will outline the means required to work with the temperature data.

**Programming languages and/or tools:**

* C/C++ compiler (informally known as ‘Arduino language’) along with both Arduino circuit board and open-source Integrated Development Environment (IDE)
* Python 3.6 via PyCharm CE open-source IDE
* MacBook Pro – macOS Sierra Version 10.12.6
* Microsoft Excel Macro and VBA Editor
* PostgreSQL via open-source pgAdmin3

To commence this project, a temperature sensor was connected to an Arduino board and programmed using C/C++ throughout Arduino’s open source IDE. However, these devices are required to get physically setup by following the connection schematics, also known as electric diagrams. These diagrams or instructions vary in the complexity of the devices that you are building. For the sake of simplicity, this paper will not focus on in-depth aspects of electrical engineering and/or the principles of electricity however I decided to provide the following picture (*Elegoo*, 105), which describes how a thermometer needs to be plugged:



Once the device is ready and connected (via USB) to the computer, we need to make sure throughout Arduino’s IDE that our script (file with .ino extension) is debugged and that the code lines match with the diagram. By doing so, the appropriate amount of voltage or electric energy will be able to power up the circuit and commence sending or displaying live temperature data per second. In addition, a Liquid Crystal Display (LCD) screen was connected to the device so that we can confirm that the sensor is in fact reading the room temperature. We can also prove that our digital thermometer is working by touching the sensor directly; since human body temperature is greater, we will notice that readings will increase immediately. However, to avoid altering the results, touching or testing the sensor was perform prior to recording our room temperature readings. Although not required for the thermometer to run, the LCD is useful for being able to compare results directly from the device with those on being displayed on the Python 3.6 environment or IDE.

After this point, another script or code written in Python will get ran so a direct connection will be established with the thermometer via USB. The Python code will receive and display the data at the same rate as the Arduino script, meaning that both codes will be running at the same time. However, the Python script will not only display the data point as temperature in both Celsius and Fahrenheit with two decimal places but will separate these by a comma, and then it will assign a timestamp to each record every second. The timestamp will get split into date and time, and then separated by a comma as well. In this case, the Arduino device cannot be programmed to assign date and time to each data point. Another function my Python code does, is to extract the data being read and convert into readable information for Microsoft Excel. For instance, if the room temperature during one second was 70º, that record would be “21.47,70.64,05/28/2019,17:59:05”. The intention of including two decimals to the temperature is to increase the precision of our data, consequently quality as well. Please note that the first number listed in the parenthesis mentioned previously refers to Celsius and the second to Fahrenheit. In addition, each record must have the same format or pattern as the previous example and any string of data that doesn’t meet the format requirements will not get displayed. Due to an unknown reason, there were a few or rare occasions in which readings from the sensor showed up as bizarre symbols or numbers, meaning that the Python script required to get a conditional statement that would allow any readings not following the format to get excluded. Basically, each record needs to have temperature, date and time separated by a comma and the same length or number of characters. This is essential since commas are required so that we can define parameters and set delimiters as we import raw data into Excel.

As mentioned previously, temperature information along with the date and time get displayed in the PyCharm IDE/platform. This happens during the moment in which the digital thermometer gets turned on and the script gets ran on PyCharm during the same time. We will also stop receiving data anytime the code gets interrupted or if the thermometer gets unplugged. Anyhow, the data being shown in PyCharm will also be the same as the information being appended to our ‘output’ file titled *Arduino\_Temperature\_Raw\_Data.txt*. This resulting text file is basically a list where each row will be used later in Microsoft Excel. Please note that I decided to name this file as such since the naming convention is the most practical for one data set (equivalent to two day-worth), however this text file can be named in anyway. If that wasn’t the case then the naming convention of the output file(s) would be similar but with a date attached to it so that each text file can be easily distinguished. After the output text file containing the sensor’s raw temperature data is ready with the amount of information we desire, then we can import it to Excel.

As a reminder, the text file containing raw data is simply a list in which each row represents a record with commas serving as the delimiters, which allow us to distinguish which characters or numbers represent temperature, date and time. The second reason why we require temperature data to have this format or structure is for us to record an Excel Macro or alternatively write or create an Excel VBA script.

This process was done the first time by opening a new excel file, clicking where it says *Record Macro* under the Developer section, manually modifying the raw data and finally stop recording once the steps were completed. Since the macro will mimic the steps performed by the user, it will be programmed to automatically import the output text file (created by the Python script), and then divide each record into four components (temperature in Celsius, temperature in Fahrenheit, date in mm/dd/yyyy format and time in hh/mm/ss format). After splitting each record into four sections, four columns in the Excel spreadsheet will be produced. Then the macro will add a total of four headers in the first row of the spreadsheet and finally insert a filter to these. The titles for each column or header names are the following: Temperature (ºC), Temperature (ºF), Date and Time. So far, the only issue I have encountered that was potentially affecting the quality of the processed data was blank spaces being created at the end of each value inside each cell within the Time column. The reason why this is a problem is due that it was causing the Time column to no longer have a valid or a recognized (by Microsoft Excel) time format. I was not able to find an explanation of why my Python script was creating one space at the right of each time value, for instance the produces value was “17:59:05 ” instead of “17:59:05”. However, instead of spending extra time on modifying my Python code, I decided that the most practical solution, while using Excel, would be to systematically replace any blank space with *no blank* spaces. Once this issue was solved, the macro was setup to stop recording since its function was completed. The macro and the Excel file were saved afterwards in xlsm format.

In addition, in case that new data were to get appended or included to the output text file, the macro will update the results every time it gets ran. If an update was necessary, the macro could be run in any new or existing Excel workbook, however the original file containing the macro is required to be opened at the same time as well. When that happened, the macro was available in the workbook’s Developer section after clicking *Macros*. If the file containing the macro was close or disabled, then the name of the macro we created wouldn’t show up. Since we are only working with one text file, we don’t need to create more than one macro for the purposes of this project. At this point the raw data from the output text file (from Python) was processed and ready get loaded into the database for storage or record keeping purposes.

The final step in this project was to create a database and then a table (basically a subset of a database) so that our recorded room temperature data could get stored. However, the table titled *RECORDED\_ROOM\_TEMPERATURE* that I created throughout SQL syntax needs to meet specific requirements so that the records could be loadable. The conditions for each column in are that temperature (in Celsius and Fahrenheit) needs to be a decimal number with two places, date needs to have a recognizable format, and the time needs to be unique (meaning that you cannot have two different temperature records assigned to the same exact time); thus, it needs to have time in a recognizable format as well. After running the macro, we need to save the resulting spreadsheet as a csv file. The reason being that the SQL database created throughout pgAdmin3 does not accept any other type of file. Once the csv meets the table’s format or structure requirements, data should be able to get loaded except for the csv’s headers of course, however, these need to match the header names created for the table. The way we confirm that our data was loaded appropriately was by running a basic SQL query which selects every column matching the columns that belong to the csv file. If our results look identical to the ones in the csv, it means our records were stored correctly and without errors. Since nothing else other than room temperature was recorded during this project, no other tables containing different type of data besides the one mentioned previously were created during this project.

Room temperature was successfully obtained from a source or device, interpreted by a Python code, processed throughout an Excel Macro or VBA script, and finally stored without errors into a SQL table within a database. The goal of this project was not only to demonstrate and simulate how simple information is developed from scratch into a final product (in this case, our temperature records) throughout several steps and making use of several tools. One of the most important aspects when it comes to data manipulation is to preserve quality. To be able to get data smoothly processed and show accurate results, extra steps need to be applied whenever we encounter expected or unexpected discrepancies, such as erroneous formatting. In this case, the Arduino’s sensor data wasn’t as complex or extensive, meaning that quality is easier to maintain, although if that wasn’t the case, then higher quality checks or standards would be essential. Thank you for taking the time to read this writing sample.

References

Stack Overflow. (2018). Retrieved from https://stackoverflow.com.

Electrical Engineering. (2018). *Electronics Stack Exchange*. Retrieved from

https://electronics.stackexchange.com.

Elegoo. (2017). *Elegoo Super Starter Kit for UNO*: 104-108. Retrieved from

http://www.myqueenstown.org/pdf/Elegoo%20Super%20Starter%20Kit%20for%20UNO%20V1.0.2017.7.9.pdf.

Language Reference for Visual Basic for Applications (VBA). (2018). *Microsoft Docs*.

Retrieved from h<ttps://docs.microsoft.com/en-us/office/vba/api/overview/language-r>eference.

Stack Overflow. (2018). Retrieved from <https://stackoverflow.com>.

The PostgreSQL Global Development Group. (2016). *Documentation*. Retrieved from

https://www.postgresql.org/docs/9.1/index.html.