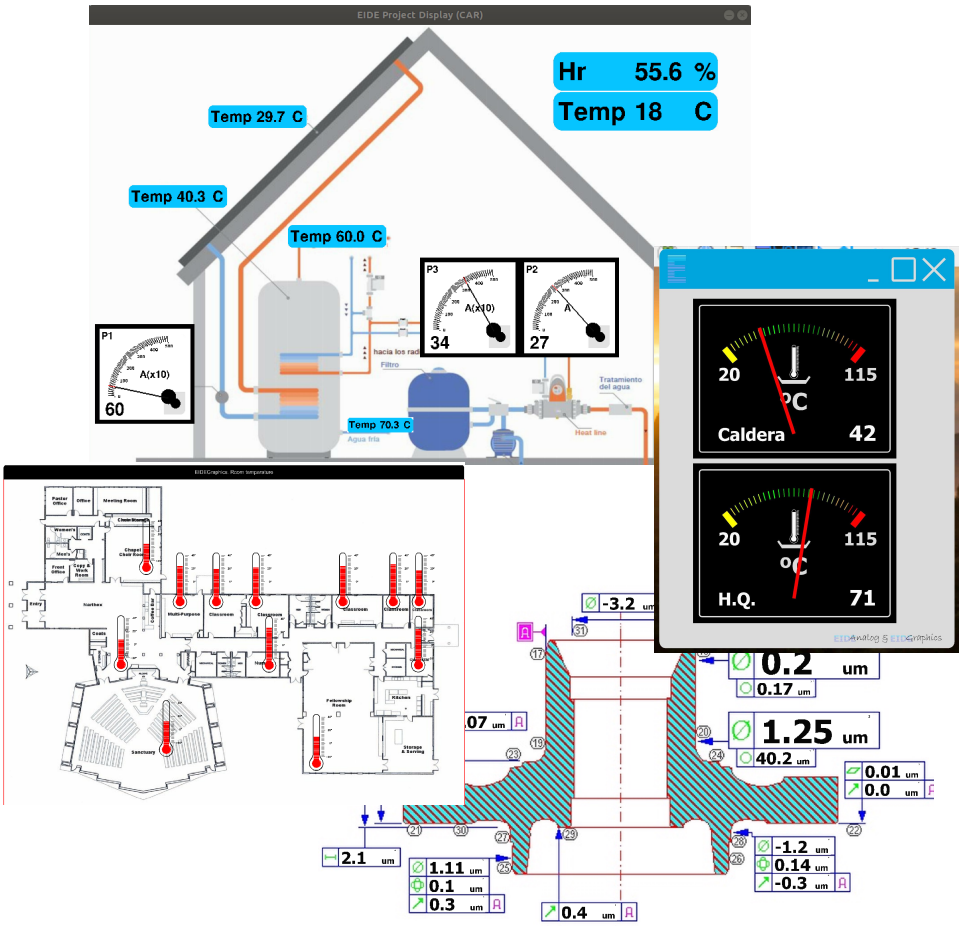
Data acquisition using Python and Raspberry Pi



EIDEAnalog library
(Making of)

Revisions

Nº	Date	Paragraph	Description
0	May 2020, the 7th	-	First chapter issued.

Table of Contents

1.- Brief introduction.

We will explain through this 'blog' how a *library* to help capturing physical parameters (analog-digital conversion) has been conceived and developed using Python as a programming language and OOP (Object Oriented Programming) as a technique. The hardware to be used is A) a Raspberry Pi card without added hardware for the oneWire bus, B) the same Raspberry plus an ADS1115 type analog to digital converter and C) the Raspberry connected to an Arduino ProMini; for the first option probes used are of the DS18B20 type, for the other two *combos* half a dozen conventional sensors are used - temperature, voltage - which will be opportunatelly depicted. As will be exhaustively explained in the blog, much emphasis is placed on the decission of the pattern of (the classes that make it up) the library and its development, issues that are usually the most complex in this type of programs.

The blog may be of interest:

- As a guide to programming practices for intermediate level courses. It may be even possible that in certain universitary degrees it will serve for the same purpose.

- For fans of (or professionals in) programming who wish to improve or test their knowledge of OOP.

- Engineers in the ‘data acquisition’ field. Although the hardware used has not an industrial grade (not, at least, the one that has been specifically used to prepare this blog), it is not ruled out that it may be of their interest.

Descriptions along the blog are intended to be as concise as the subject permits, which, as will be pointed several times, is not a simple issue.

(Should you just want to familiarize yourself with the use of the library and not with its design, please go directly to point 8, "Use of the library").

1.1.- Blog declared goal.

The stated objective of the blog is, therefore, to developp a -small- library written in Python to ease the use of various types of AD adapters, the popular ADS1115, the "oneWire" bus and an Arduino Promini. The first and the third are hardware additions to the Raspberry, while the oneWire bus is a Raspberry *native* option: loaded the driver, you can enable the pins of the GPIO connector that you deem appropiate to implement it.

Configuring the Raspberry: Depending on the version of the Raspberry you have and the Raspbian *distribution*, you will have your Raspberry configured for the i2c bus (necessary for the ADS1115 to work) and/or oneWire (necessary for the bus of the same name) and/or serial communication (necessary for the Arduino). See Annex II, “Configure your Raspbian/ Raspberry” for details.

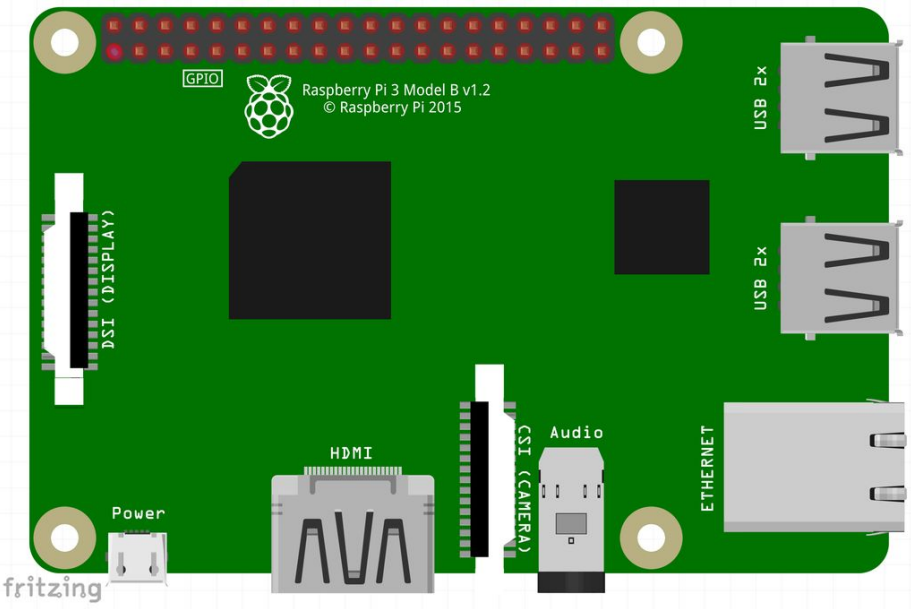


Figure 1. Raspberry 3B.

Yet, the blog is not limited to this: although the library is actually developed -as it is written- as outlined in the previous paragraph, there is another objective as important as the stated one, at least for who wants to learn some analysis and OOP programming. Let’s hope that the trip, encompassing the needs, analysis, problem decomposition into classes and the discussion of the final pattern of the library, is as useful as its -trip’s one- destination, which is the mere use of the library.

1.1.1.- OOP design pattern.

To focusing the question: when addressing an OOP development analysis, one of the main issues is the *architecture* of the system, its pattern; i.e.: How do I find the classes that will be instantiated to give rise to the objects that make up my program?. How much should I crumble -granularity- the problem?. How many classes?. What are going to be the relationships among them? And the objects among themselves? How is it documented?

The specific matter that is addressed in this blog -the library itself- is quite simple: you can solve it with hardly a dozen classes (as we will see later). Its best feature is that it is fully developed (that the library code is available https://github.com/Clave-EIDEAnalog/EIDEAnalog -and that it works!). The reader, whose knees will probably tremble sometimes, is at least sure to come to an end.

In other words, the best virtue of this blog is that it is a fully developed example that does not limit to describe the basics of the OOP analysis, a complex issue that requires much more than this blog. If you want to follow it -the blog- please keep this in mind: it is a developed (programmed, working) example. Nothing more, nothing less.

1.1.2.- Data acquisition systems. A/D conversion.

If it is a matter of doing OOP pedagogy, one could have chosen a subject other than A/D (analog to digital) conversion . What does this have -that others do not- to have been chosen as the common thread of the blog?

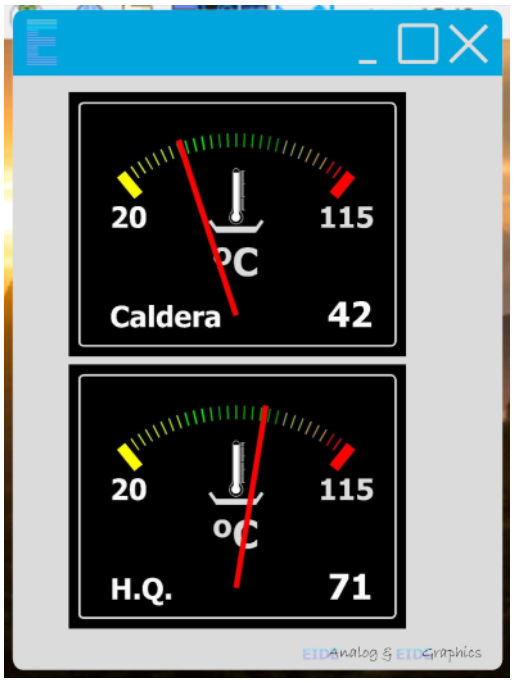


Figure 2. Raspberry data acquisition system.

That the blog authors know it well (and are perfectly aware that choosing a library to "acquire physical parameters", as stated in the third line of the first point of this blog, will

take away a good part of potential followers who, a different topic chosen, perhaps would have been encouraged to follow it).

On the other hand, may be that the matter also is atractive, as the result is connecting sensors to a computer (an issue that, in many ways, can be considered eccentric). This is an issue of interest to a good number of activities, both in the field of industry and training, and in this - that of training – to all the engineering and degrees in physical or chemical sciences. At the end of the day, and except for the subtleties of the A/D conversion device itself -which are specific to this discipline- the remaining is not much more than algebra and binary logic; once you are satisfied with the purpose of what you are doing -the library-, the way of doing it -analysis- is valid for similar problems, or of a similar approaches.

1.1.3.- Blog development. Pedagogics.

We have been careful in the writing of the blog, using a pedagogical style. It is not just a matter of writing some classes so that they can be incorporated into a project by a “copy-paste” (yet this can be done without a problem - and it works). The main goal of the blog is, as mentioned, to help anyone who wants to analyze a case on how to approach the analysis and development of a library using OOP techniques.

1.1.4.- Conclusion. Goals.

By the end of the blog you should have acquired certain skills regarding OOP (see 2.5.3, “OOP. Conclusion."). In addition you will learn:

- Computer data acquisition (basic concepts). ADC (Analog to digital converters).
- User level Raspberry management. (Very) basic concepts of Linux.
- (Rudiments of) operation of oneWire, i2c buses and serial communication (UART).
- Functioning of some commercial (and popular) sensors.
- Thermistors. NTC. Tabulated sensors.

1.2.- Basic knowledges to follow the blog. Prerequisites.

Blog authors have experienced more than once manuals that begin -enthusiastically- explaining that to use the computer mouse you have to operate it with your hand, move it across the pad and press the left button to select or the right one for options (and that can be configured the opposite for left-handed); inevitably the manual -or whatever- abandons this level of detail on the second paragraph to, much worse, give indecipherable instructions just three or four paragraphs later (each of them -of the indecipherable instructions-, deserving for themselves a manual).

We will try not to repeat this error; It would be delusional to think that in order to follow this blog you do not need any prior knowledge of the subjects being addressed. Whoever that wants to follow it must have a *user knowledge* (whatever this means) of computing, personal computers. As we use a Raspberry as hardware you should familiarize with it (of course, whenever the Raspberry is referred to, we are also including the Linux operating system).

Figure 3. ADS1115 tied to a Raspberry.

Nor you should get disappointed for getting your hands a little dirty: whatever the option you choose to implement the blog content you will have to connect something to the Raspberry -see Figure 3 above- and tighten some screws (terminals). We encourage you to connect things to each other: the Raspberry with the A/D converter, the probes to the converter, the Raspberry's power supply (it's like the one of a mobile phone), ... nothing too complicated.

Let’s go step by step.

1.2.1.- Electronics. A/D converters. Binary numbers.

You should know how to use a polymeter; it is worth knowing how to measure DC voltages. A review on Ohm's law won't hurt (although it is not estrictly needed for the blog).

Figure 4. Polymeter.

The ADC (analog to digital conversion) issue is a bit trickier.

When it comes to acquiring values *quickly*, the A/D conversion issue is truly a challenging one. *Quickly* means thousands of times per second -or more; or much more- which is the case, for example, on an audio studio to capture music without losing quality (44,100 times per second); similar speeds are required in many problems in the industry of all kinds, in research, telecommunications, image (much more, this case).

This is not this blog case; it is enough understanding that it is all about converting real world parameters to numbers much more calmly. Although we will have to delve into how the A/D converter is controlled, this is a more *administrative* than an engineering problem, as we will see when it comes to it.

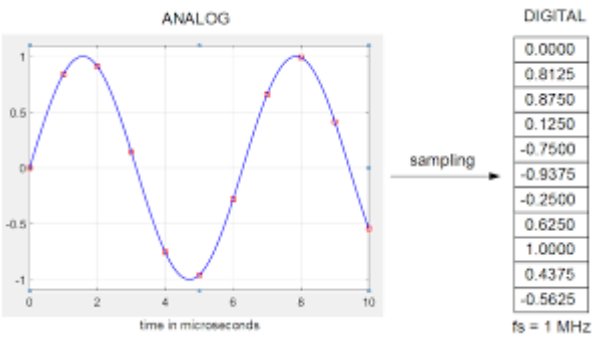


Figure 5. A/D conversion.

(As some WEB page explains, you sitting next to a mercury column thermometer taking notes every, say, 10 seconds, becomes an analog-to-digital converter: you are transforming -and recording- analog values to numbers).

The point is that the computer works with the base 2 numbering system (the -in-famous 1010001010 ...), which in this blog becomes the main scientific problem (at least different from the basic one: the analysis in OOP). If you already master this matter, congratulations, please go to the next paragraph; should you not, you have to make some investigation: entries of wikipedia for "*binary*" and "*bitwise*" are a good start. You also have to know what the hell the "two's complement" format is.

A basic knowledge of the terms *byte* and *word* are necessary too.

Incidentally, the hexadecimal numbering system is also mentioned; take a look at it on wikipedia.

1.2.2.- Raspberry.

Everything you need to know about the Raspberry hardware, at least to start with, is 1) how to connect it and 2) how to configure the µSD (microSD). There are literally thousands of WEBs that explain how to connect it; even the accompanying leaflet includes it.

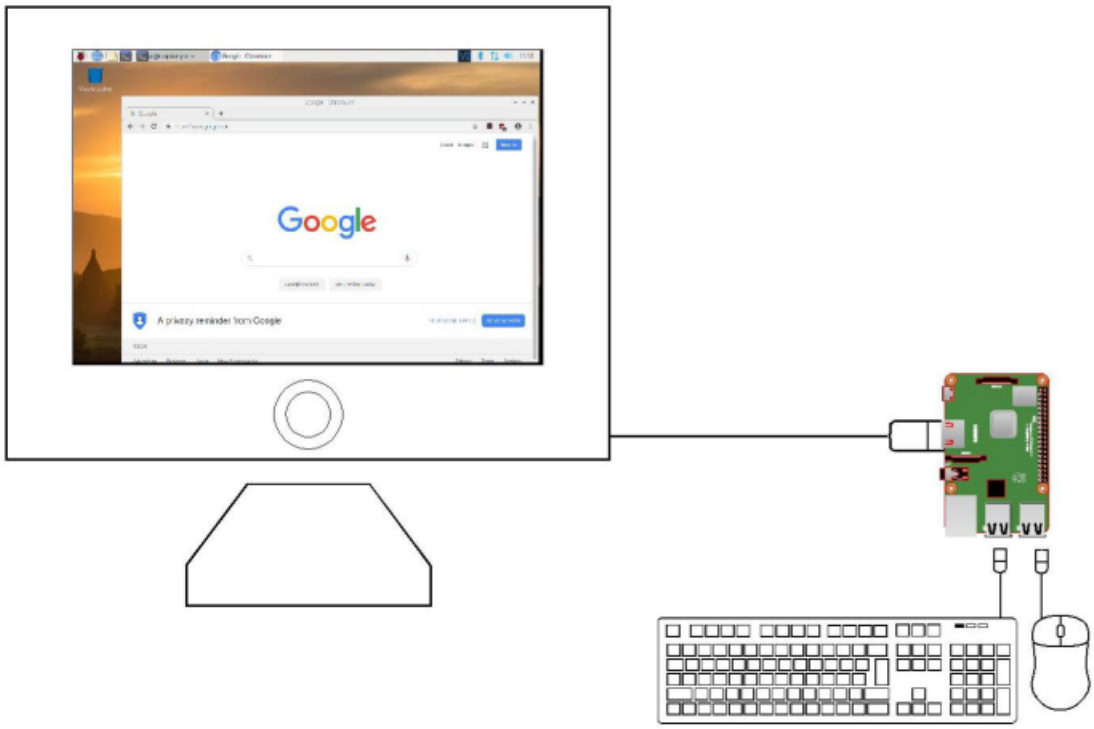


Figure 6. A Raspberry acting as a laptop.

(The easy way to get started is to connect an HDMI monitor to the video port, a keyboard and a USB mouse to two of the 4 USB ports it has, insert a properly configured µSD memory card and power it all.).

The µSD thing is a bit harder: see Annex II, “Raspbian Configuration” for more information.

Although as we present the different subjects they will be explained in detail, it does not hurt finding out what a oneWire, an i2c and a serial communication buses are. Take a look at the wikipedia entries for “oneWire”, “i2c” and “Serial Communication”.

1.2.3.- Linux.

The Raspberry has the enormous advantage of working with Linux ... should you not know Linux you miss it. But do not panic, it is not necessary to become an expert to use it, and using the graphical interface is quite simple to start with.

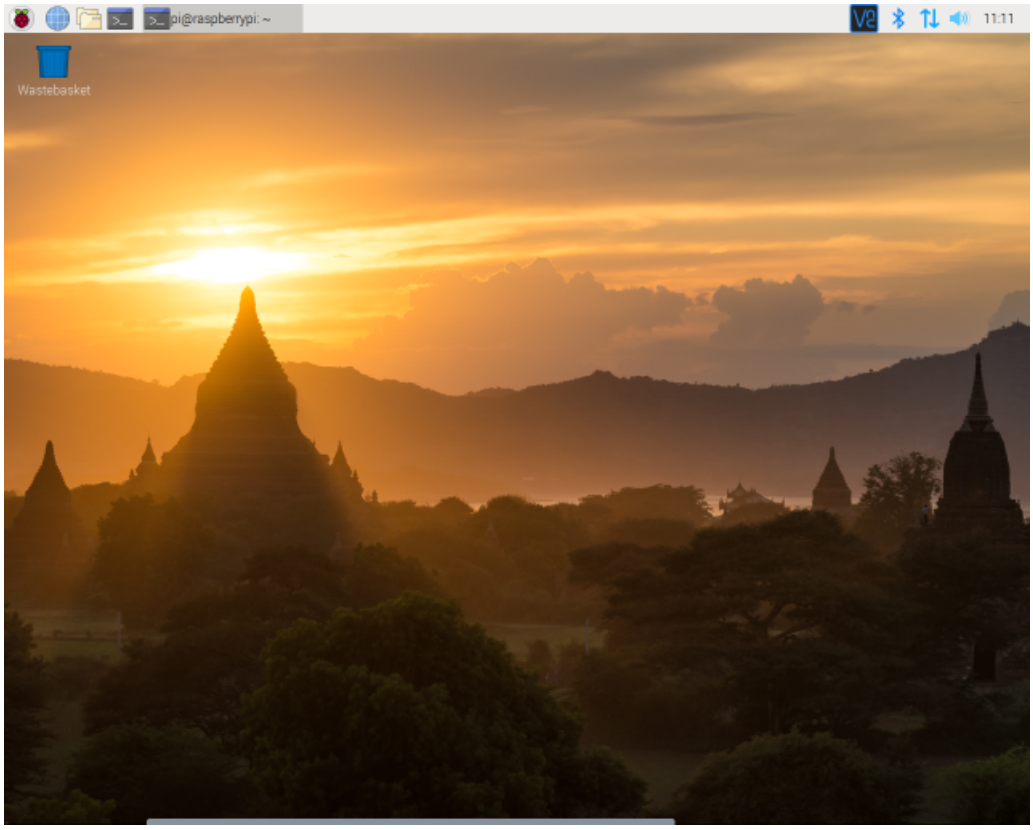


Figure 7. Linux interface (Raspbian).

To following the blog, you don't have to know much about Linux: should you have a well configured *distribution*, a user knowledge should be enough to start (see annex II, “Raspbian configuration”).

As in the precedent paragraph, if you manage with Linux, congratulations, please go to the next paragraph.

Else, we recommend you to learn a bit:

- It is in your best interest to read something to at least familiarize yourself with the terms ("Debian", "Jessie", "Raspbian", "Noobs", "Ubuntu", ... don't be alarmed, they are very much the same). Read the article from wikipedia for “Raspbian” (be patient: you won't understand many things at first, but you have to get a bit comfortable with them); Raspberry's website (“www.raspberry.org”) is not the best in the world, but having a look at it is a good idea either.

- “Terminal": the term refers to the use of the operating system (Linux, in this case, although it can be applied to any O.S.) by means of a "command line". The elders in the city would remember the black screen of MS-DOS prior to Windows: that is the

"terminal " (and, by the way, a few MS-DOS commands are copied from Linux; they are the same).

- It is quite convenient that you know some commands to configure Raspberry things (again the reader is referred to Annex II, “Raspbian Configuration”). See
  - https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners
  - https://www.digitalocean.com/community/tutorials/an-introduction-to-linux-basics
  - https://www.freecodecamp.org/news/the-best-linux-tutorials/
  - https://www.tutorialspoint.com/unix/
  - The book “The Linux Command Line: A Complete Introduction. William E. Shotts Jr ” is excellent.

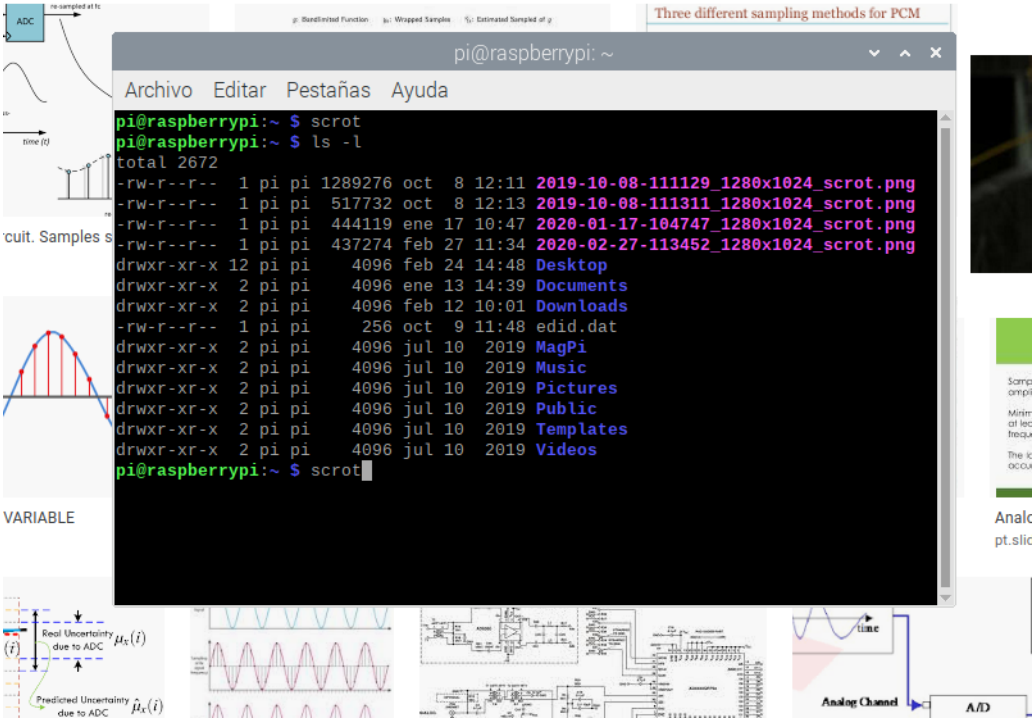


Figure 7. Linux terminal session (Raspbian).

Backup copies: One of the main advantages of Linux is the ability of recovering from the disaster of the loss of the S.O. itself. Using *another* operating system, the hard drive failure leaves you out in the open. In the case of the Raspberry, the equivalent would be the failure of the µSD (or the Raspberry itself, the worst disaster); The good -excellent- new is that it is possible to have a copy of the µSD simply by using another µSD. If you are going to mess around with the Raspberry, which we recommend, it is convenient that you have an updated backup by making periodic copies from one µSD onto the other: a few dollars in exchange of a priceless tranquility.

1.2.4.- Python. OOP.

This blog is not a Python tutorial (nor an OOP one). To follow it without too many shocks -without having to return frequently to more basic issues than those required to understand what is being developed- it is necessary to have a reasonable knowledge of Python. And you have to have written half a dozen simple Python programs.

(The term "reasonable" in the previous paragraph is used with some bitterness because, who is able to say what a *reasonable* knowledge of Python is?).

Python is one of the easiest (the easiest?) programming languages to learn, and it is extremely powerful. And it is not just that its syntax allows coding in a way that summarizes in a single line of, say, 30 or 40 characters what using other languages requires many lines: Python has implemented as native all the instructions and architecture of any other modern language, and you have to take into account what is added by the libraries included in the normal download. It also has structures -dictionaries, for instance- and/or novelties -class attributes- that at the time, and still today, makes it different from the closer environment.

Python is, in summary, a world to explore. So let's see who is the brave woman or man who defines what a *reasonable* knowledge of Python is.

So we will proceed by extension: let’s take the Python WEB reference tutorial (“https://docs.Python.org/3/tutorial”) as a guide. You need to know:

- How to create and manage numeric variables, their types -int(eger), float, ..- and formats -decimal, hexadecimal, binary, ...-. All the arithmetic and logical operations. You need to understand what “None” is.

- How to create and manage strings (the basics: what a string is).

- "Flow control". The above mentioned Python tutorial is not among the best of the foundation (“Python Software Foundation”): the first mention it makes on this concern -flow control of a program- in point 3.2 is undoubtely unfortunate: one cannot understand why the –very first- example is implemented with a *while* without any previous anesthesia. To follow the blog smoothly go to point “4”: you have to understand all the contents in chapters 4.1 to 4.6, all.

- Lists, tuples and dictionaries: in depth. You should be familiar with all the content of tutorial points 5.1.1, 5.1.2. and 5.5. Along the library they are used a lot to iterate: please study in depth the content of point 5.6 and review the point 4.2.

- Modules: you need to understand what the "import" instruction is for and what is meant by "standard modules" (6.2).

- **Errors and exceptions**. They deserve thir own comment: the implementation of a library that is worth of that name should not be made available to potential users without it having a minimum of *protection* against the mistakes that will surely be made when using it (and the ones that, undoubtedly, will be utterly hidden in the code of the library itself). The counterpart is that the use of error handling clauses

introduces a new complication in the ‘aspect’ of the code, which, at least at first glance, causes the reject of rookies.

A necessary compromise has been reached: the first version of the library (the one that is discussed in depth into this blog and can be downloaded from https://github.com/Clave-EIDEAnalog/EIDEAnalog), has no *error handling* code. A ‘Pro’ version that will have it (open source too) is in preparation. Have a look on chapter 8 of the tutorial, though.

- Classes. Objects: This is the moment of the truth. The terms *class* and *object* should have not secrets for you; and we are off to a bad start, however, if you frown as soon as the first *self* appears or you panic when you see a *__init__* coming floating in the code. Along the blog we neither take everything for granted (related to the concepts of class, object, etc.) nor can we make pedagogy about it with the basic concepts of the subject (which, on the other hand, is done -pedagogy- when the things get complicated: abstraction, encapsulation, inheritance, polymorphism, patterns). On the other hand, the didactic standard of the chapter 9 of the above mentioned tutorial (“https://docs.Python.org/3/tutorial”) leaves, again, something to be desired. And this is not an easy matter (neither the basic question of the OOP technique nor how to start with it). Please, find below a couple of WEB sites that you should study if you want to follow the blog contents without shocks.
    - https://www.programiz.com/Python-programming
    - https://realpython.com

Python download, install and configuration: If you are not familiar with Linux, it is possible that starting up Python on Raspberry involves some difficulty. Although the authors of the blog are unrepentant supporters of Linux -and, therefore, self-conscious detractors of the "other" operating system-, it would be an excess to suggest to the blog follower to study it thoroughly -Linux- as a previous step to start with the blog . The Raspberry startup and use is not complicated at all if you have a good *distribution* (the jargon to refer to the version of the S.O.). There are many vendors on the Internet who can supply you with a properly configured µSD. Linux has a graphical interface similar to that of Windows. See Annex II, “Raspbian Configuration” for complete information.

1.2.5.- Other prerequisites.

- File types: You have to be able to identify and modify a text file (windows *notepad*; Linux *leafpad*).
- Needless to say that you have to know what the ASCII code is.
- Have a look into the wikipedia entrances for NTC and thermistor.