

## Chapter 6 Raspberry Pi and grovepi with Python: Sensors, actuators and interfaces

### Capítulo 6 Raspberry Pi y grovepi con Python: Sensores, actuadores e interfaces

CORONA-SÁNCHEZ, Ernesto<sup>\*1</sup>, NAMIGTLE-JIMENEZ, Jesús<sup>1,2</sup>, MASTACHE-MASTACHE, Jorge Edmundo<sup>1,2</sup> and LÓPEZ-RAMÍREZ, Roberto<sup>2</sup>

<sup>1</sup>Universidad de Ixtlahuaca, Facultad de Ingeniería

<sup>2</sup>TecNM/Tecnológico de Estudios Superiores de Jocotitlán

ID 1<sup>st</sup> Author: *Ernesto, Corona-Sánchez* / **ORC ID:** 0009-0004-8828-3400

ID 1<sup>st</sup> Co-author: *Jesús, Namigtle-Jiménez* / **ORC ID:** 0000-0002-0908-4592, **CVU CONAHCYT ID:** 624757

ID 2<sup>nd</sup> Co-author: *Jorge Edmundo, Mastache-Mastache* / **ORC ID:** 0000-0001-6104-6764, **CVU CONAHCYT ID:** 544943

ID 3<sup>rd</sup> Co-author: *Roberto, López-Ramírez* / **ORC ID:** 0000-0001-8341-3684, **CVU CONAHCYT ID:** 233228

**DOI:** 10.35429/H.2023.5.86.110

E. Corona, J. Namigtle, J. Mastache and R. López

\*jesus.namigtle@tesjo.edu.mx

R. López (AA.) Engineering and Architecture in the Northern part of the State of Mexico. Handbooks-TI-©ECORFAN-Mexico, Estado de México, 2023

## Abstract

Currently we have seen the launching and distribution of a wide variety of programmable boards, from the well-known Arduino to boards such as Raspberry Pi, Orange Pi, SparkFun RedBoard Artemis, among others. Each one of them presents differences not only in design but also in the language used to work with them, memory capacity, processor type, etc. Another characteristic that distinguishes them is in the extensions and complements such as hats, sensors and actuators that enhance their use. The cards are a fundamental element in the development of projects focused on robotics, monitoring systems, IoT technology applications to name a few, which is why this chapter will delve into one of the extensions compatible with Raspberry Pi, GrovePi+. This add-on presents advantages when using humidity, temperature, light, air quality sensors, as well as actuators of different types. In addition, graphical interfaces will be implemented taking advantage of the Python3 distribution available in Raspberry Pi; in this way, elements that maximize the use of the hat can be used. This with the purpose that the user can understand and manipulate the types of ports available (analog and digital mainly), presenting the Grove family modules from a didactic point of view.

## Monitoring, GrovePi+, Raspberry Pi, Interfaces, Programming

### Resumen

Actualmente se ha visto el lanzamiento y distribución de una gran variedad de tarjetas programables, desde el reconocido Arduino hasta tarjetas como Raspberry Pi, Orange Pi, SparkFun RedBoard Artemis, entre otras. Cada una de ellas presenta diferencias no sólo en el diseño sino también en el lenguaje utilizado para trabajar con ellas, capacidad de memoria, tipo de procesador, etc. Otra característica que las distinguen está en las extensiones y complementos como hats, sensores y actuadores que potencian su uso. Las tarjetas son un elemento fundamental en el desarrollo de proyectos enfocados en robótica, sistemas de monitoreo, aplicaciones tecnológicas IoT por mencionar algunas, es por ello que en este capítulo se profundizará en una de las extensiones compatibles con Raspberry Pi, GrovePi+. Este complemento presenta ventajas a la hora de utilizar sensores de humedad, temperatura, luz, calidad del aire, así como actuadores de distinto tipo. Además, se implementarán interfaces gráficas aprovechando que la distribución Python3 está disponible en Raspberry Pi; de esta forma se pueden utilizar elementos que maximicen el uso del hat. Esto con la finalidad de que el usuario pueda entender y manipular los tipos de puertos disponibles (analógicos y digitales principalmente), presentando los módulos de la familia Grove desde un punto de vista didáctico.

## Monitoreo, GrovePi+, Raspberry Pi, Interfaces, Programación

### 1 Introduction

Over the years, research in different areas of engineering has presented systems in which a programmable card is indispensable, either for data acquisition and processing purposes, for a control phase, as a means of communication with the cloud, etc. However, several cards have a high cost in the market and come to need extensions, devices or add-ons that require specific libraries, present incompatibility or simply consume a high amount of resources being undesirable the addition of new components. Therefore, GrovePi+ being a Raspberry Pi compatible add-on can be a good option to work with modules of different functions without exceeding the resources it has. It is a hat that provides analog, digital and serial ports that are used to connect basic electronic components, as well as actuators linked to a single Python-based library for manipulation. The purpose of this work is to show the user how to use some elements through simple code segments providing enough tools to make small control systems.

### Short introduction to Raspberry Pi

A Raspberry pi can be considered from the user's point of view, a kind of mini-computer that allows you to perform various projects thanks to the functions it can provide. It consists of a board, a processor, different types of pins and connection ports. This board, launched by the Raspberry Pi foundation (its first model) in 2012 has had variants over time, however, it can be narrowed down into 2 models; A and B. You can also find other models that although they can be differentiated by the size or the addition of its keyboard (as for example the Raspberry Pi 400) maintain the same architecture in their models A and / or B.

Raspberry Pi OS or also known as Raspbian is the official operating system of the board. This Linux distro (Linux distribution) is based on Debian, optimized to run on ARM hardware.

**Figure 1** Raspberry Pi 3B+ (Kurniawan A.)



### Raspberry Pi hardware

The models that provide the most features to the user are models A and B, and the main difference between them is in the USB connection. Model A versions consume less power and do not have an Ethernet port, unlike model B versions which do.

A standard Raspberry board consists of the following components:

- RAM memory.
- Processor(CPU)
- Graphics Processor (GPU)
- Ethernet connection.
- GPIO interface
- XBEE socket (wireless communication)
- UART (Serial Interface)
- Power supply connection.
- Connection for external hardware (microSD memory)

Where, the microSD memory is necessary for storage and booting of the Pi board. From the point of view of a Windows user the microSD would be the hard disk of the PC. Los Distintos Modelos de Raspberry Pi.

Table 1 presents a summary of the various versions and models of the Raspberry Pi board, highlighting the improvement that has brought model after model, positioning this board as one of the most versatile and economical in the market.

Table 1 Raspberry Pi boards comparison

Model	RPI 3	RPI2	RPI B+	RPI A+	RPI ZERO	RPI B	COMPUTE
Characteristics	Performance /Wi-Fi /Bluetooth/ Ethernet	Performance /Ethernet	Ethernet	Price	Price /Size	Original	Integration /eMMC
Price	\$35	\$35	\$25	\$20	\$5+	\$25	\$40
Processor*	BCM2837 quad core Linux ARMv7	BCM2836 quad core Linux ARMv7	BCM2835 Linux ARMv6	BCM2835 Linux ARMv6	BCM2835 Linux ARMv6	BCM2835 Linux ARMv6	BCM2835 Linux ARMv6
Speed	1.2GHz	500MHz	700MHz	700MHz	1GHz	700MHz	700MHz
Memory	1GB	1GB	512MB	256MB	512MB	512MB	512MB
Typical Power	2.5W (up to 6.5W)	2.5W (up to 4.1W)	1W (up to 1.5W)	1W (up to 1.5W)	1W (up to 1.5W)	1W (up to 1.5W)	1W (up to 1.5W)
USB Ports	4	4	4	1	1OTG	2	via header
Ethernet	10/100 Mbps Wi-Fi and Bluetooth	10/100 Mbps	10/100 Mbps	none	none	10/100 Mbps	none
Storage	micro-SD	micro-SD	micro-SD	micro-SD	micro-SD	SO	4GB eMMC
Video	HDMI Composite	HDMI Composite	HDMI Composite	HDMI Composite	mini-HDMI composite	HDMI RCA video	HDMI via edge TV DAC via edge
Audio	HDMI digital audio and analog stereo via a 3.5mm Jack (where available)						
GPU	Dual Core VideoCore TV Multimedia Co-Processor at 250MHz (24 GFLOPS)						
Camera (CSI)	yes	yes	yes	yes	no	yes	CSI x 2 via edge
Display (DSI)	yes	yes	yes	yes	no	yes	DSI X 2 via edge
GPIO header	40 pins	40 pins	40 pins	40 pins	40 pins	26 pins	48 pins via edge
Usage	General-purpose computing and networking. High-performance interfacing. Video streaming	General-purpose computing. High-performance interfacing. Video streaming	General-purpose computing. Internet connected host. Video streaming.	Low cost general-purpose computing. Standalone electronics interfacing applications.	Low cost small profile standalone electronics interfacing projects.	General-purpose legacy applications. Internet connected host.	Suitable for plugin into user-created PCB's using a DDR2 SODIMM connector. Open-source breakout board available.

(Molloy, D., 2019)

## Raspbian for robots

*Raspbian for Robots* is a Raspbian-based operating system created by the company Dexter Industries, intended for Raspberry Pi-based robot kits. This OS contains the embedded software to connect to GoPiGo, BrickPi, GrovePi or Arduberry and program. Programming environments such as Scratch and Python are already available with several test scripts ready to just run.

## Grovepi+

Grove Pi+ is either an add-on or a frame/hat designed for the Raspberry Pi board, where the communication between the two is via an I2C interface. All GrovePi+ modules connect to universal Grove connectors consisting of 4-pin cables. Functional Grove modules with analog and digital signals are connected directly to the ATMEGA328 microcontroller in GrovePi. The microcontroller acts as an interpreter between the Raspberry Pi and the Grove sensors that send, receive and execute commands sent by Raspberry Pi.

In addition, Grove Pi allows Raspberry Pi to directly access some Grove sensors since Raspberry Pi has an I2C bus and a serial bus. The latter can be directly connected to the sensors through the I2C ports and the USART port.

This is a hat that has gained relevance in recent years due to its use in various projects such as weather stations (Bell, C., 2021) whose main modules are based on the Grove family of sensors from seeed studio. Among other noteworthy applications is the air quality monitoring system implemented with IoT (Balasubramaniyan, C., 2016), a project that managed to effectively monitor the air remotely and thus raise the possibility of covering a larger area in future work.

In the different projects that have been carried out with this hat, the ease with which the different Grove modules are connected and managed has been observed. Deepening in the code part, Python turns out to be the language with which it can be better interacted allowing that it does not have difficulties with the handling of sensors and actuators.

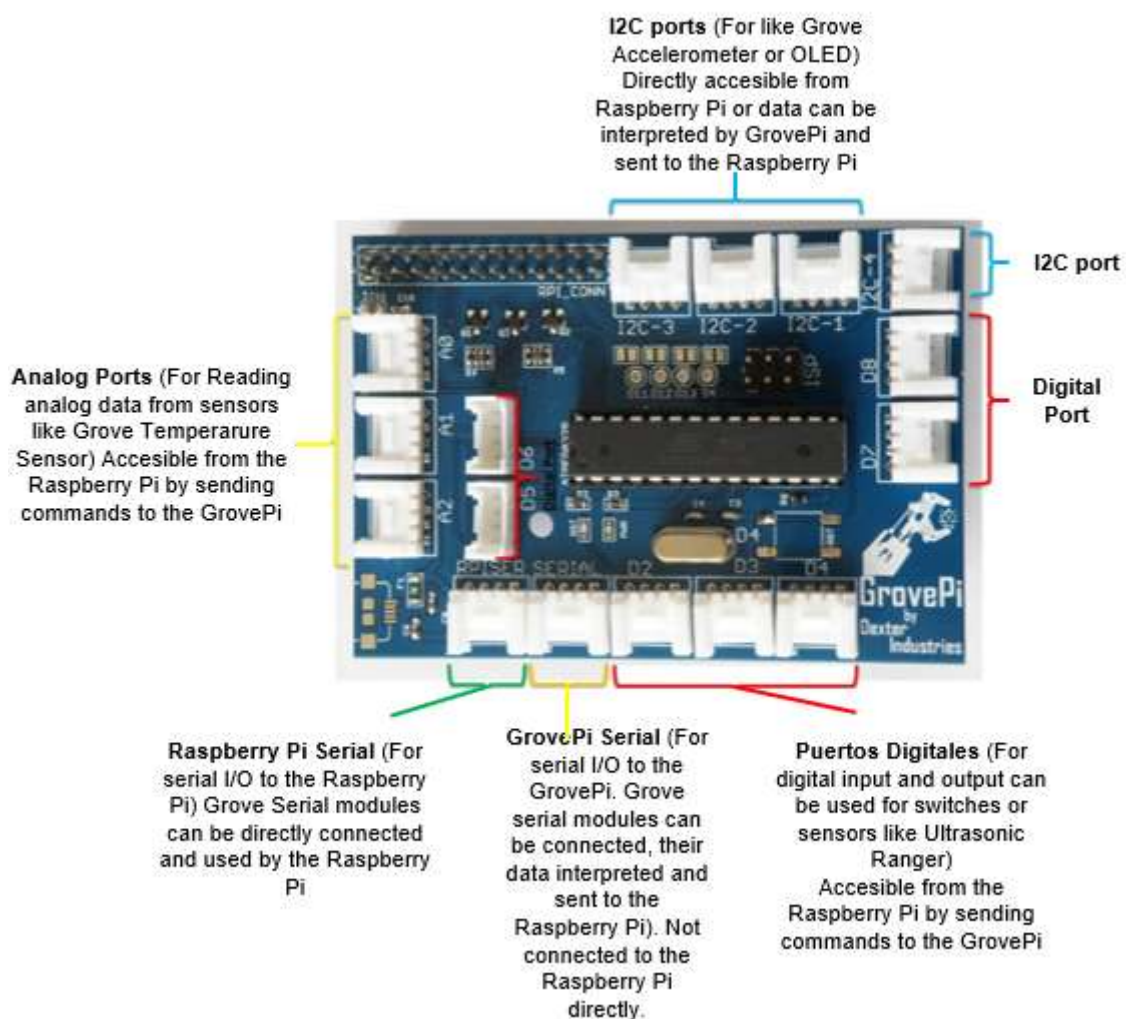
It is important to emphasize that it is thanks to the Raspbian for Robots distribution that it is possible to use GrovePi+, since it is impossible to download the repository containing the dependencies and libraries for its use from other distributions.

### GrovePi+ Port Layout

Image 2 shows the way the ports are distributed and the type of each one of them. This hat provides digital (7), analog (3), I2C (3), Grove serial (1), Raspberry Pi serial (1) ports in addition to the power (5v) and ground (GND) pins.

Between the ports already mentioned you can also find 2 rows of pins (26), these pins can be used normally as if you were only working with Raspberry Pi, so you get the most out of both Raspberry and Grove Pi+ pins.

**Figure 2** GrovePi+ ports (Dexter Industries)



## Python

Raspberry Pi has generally been seen to be used in applications based on Python, an open source programming language with a short history when talking about GrovePi+.

### Brief historical context

Python is a programming language widely used in web applications, software development, data science and machine learning (ML). Some of the reasons for using it are its efficiency and ease of learning, in addition to being able to run on several different platforms. Created by Guido Van Rossum in 1989 and released its first version in 1991 (Python 0.9.0), it included notable features such as some data types and error handling functions.

The second version (Python 1.0), released in 1994, contained new functions related to data list processing, assignment, filtering and reduction. This was followed by Python 2.0 in 2000, which provided the programmer with support for Unicode characters and a shorter way to traverse a list. The third version (Python 3.0) was released in December 2008, providing among the most outstanding improvements the print function, support for number division and error handling.

It is a language that has not yet been fully exploited in addition to having a rising popularity in recent years, and it is not for less, since it has been used in various projects involved with IoT. With this, development boards based on this language have raised alternatives according to the needs of users.

Among the works that relate the use of the language, we can find the guide for the management of programmable cards (Gonzales, R., 2021) supported by JavaScript, development and management of prototypes such as the FoamPi (Wright, H. C., Cameron, D. D., & Ryan, A. J.) whose objective is temperature monitoring of chemicals and even, it has been implemented in the field of environmental engineering by developing a software assembled in Python (Mg, I. Y. M. M. L. 2019) that helped to maintain the level of health in the ecosystem studied using the Tennat methodology. So, taking into account this kind of 'projects, Python still has a lot to offer in different research areas.

### Main characteristics of the language

Python is an interpreted language, that is, it directly executes the code line by line. If there are errors in the program, its execution is stopped. This makes it easier for the programmer to find irregularities. It is a dynamically typed language, so it is not necessary to define the type of variable when writing the code since Python automatically determines what type it refers to at runtime. It is a high-level language, so the programmer does not have to worry about its underlying functionalities such as architecture and memory management. Object-oriented, Python considers everything as an object, but it can also support other types of programming.

## IDE Thonny

Thonny is one of the IDEs originally born to program in Python on a computer that has been gradually incorporating support for Micropython. It allows working with boards such as Micro Bit, and those containing the ESP8266 and ESP32. It can also be used to program in Circuit Python.

This environment is also very easy to use and has more options and tools than other IDE's, although many of them work only when using the PC version of Python.

A very interesting feature included in the latest version is the possibility to record and update the board firmware internally from Thonny without using ESPTOOL.

It is an open source development created at the University of Tartu in Estonia and is very active, so it will surely continue to introduce new features as time goes by. It is available for Windows, Mac and Linux.

Talking a little bit about the workspace view, in *image 3* you can see the basic elements that compose the development environment. In this view you can locate the main icons; button to create a new program, open files, save button, script execution, debug button and sub-functions (skip, enter, exit and resume) as well as the stop/restart button.

In addition to the submenus (file, edit, view, run, tools and help), Thonny provides a wizard that helps the user to improve the structure of the scripts he develops, providing recommendations that can improve the debugging of the program.

The console is fundamental in environments intended to execute programming codes and Thonny is not far behind, as can be seen in *image 3*.

**Figure 3** Thonny development environment



## GrovePi

In order to work with the GrovePi+ hat it is necessary to install an exclusive library for it. Using the Raspberry Pi command terminal.

This library is mainly used for handling sensors, since GrovePi+ has several ports it is necessary to install it to be able to correctly manipulate each one of them.

Initially, the GrovePi+ shell must be placed on the Raspberry Pi. Then start the command terminal and type the *command 1*. This command creates a folder where the library will be stored.

```
mkdir ~/Dexter
```

*Command 1. Creation of main folder*

The newly created folder is then entered using the following line as specified in *command 2*.

```
cd /home/pi/Dexter
```

*Command 2. Access to the newly created folder*

For the next step you must have an active connection to an internet network, since the GrovePi repository provided by Git counter in *command 3* will be downloaded. When the download is finished, a new folder named GrovePi will be created.

```
git clone https://github.com/DexterInd/GrovePi
```

*Command 3. Download from grovepi repository*



Then, the "Script" folder is accessed as shown in *command 4*.

```
cd /home/pi/Dexter/GrovePi/Script
```

*Command 4. Access to the script folder.*

The next step is to start the installation script with the *command 5*, where the packages used by GrovePi+ will be downloaded, confirm by pressing "y" and wait for the end of the download. At the end of the download the Raspberry Pi will restart.

```
bash ./update_grovepi.sh
```

*Command 5. Installation of dependencies*

## Installation

To check that the script was installed correctly you must check if the Raspberry Pi is able to detect the GrovePi+ hat by typing the *command 6*.

```
sudo i2cdetect -y 1
```

*Command 6. Check*

The way to check that the installation was successful is to observe an "04" in the output as shown in *image 4*. The grovepi library is essential because it is through it that the GrovePi+ ports can be used, allowing analog and digital readings, serial communications, and thus take advantage of the Python language with instructions that are very reminiscent of the Arduino syntax.

**Figure 4** GrovePi+ detection

```

pi@dex: ~/Desktop/GrovePi/Script
File Edit Tabs Help
Best match: cffi 1.11.5
Processing cffi-1.11.5-py3.5-linux-armv7l.egg
cffi 1.11.5 is already the active version in easy-install.pth

Using /usr/local/lib/python3.5/dist-packages/cffi-1.11.5-py3.5-linux-armv7l.egg
Searching for pycparser==2.18
Best match: pycparser 2.18
Processing pycparser-2.18-py3.5.egg
pycparser 2.18 is already the active version in easy-install.pth

Using /usr/local/lib/python3.5/dist-packages/pycparser-2.18-py3.5.egg
Finished processing dependencies for grovepi==1.4.1
pi@dex:~/Desktop/GrovePi/Script $ sudo i2cdetect -y 1
sudo: i2cdetect: command not found
pi@dex:~/Desktop/GrovePi/Script $ sudo i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- 04  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
pi@dex:~/Desktop/GrovePi/Script $

```

## Tkinter

Tkinter is a library that provides the user with a platform-independent set of windowing tools. It uses the tkinter package and its extension tkinter.tix in addition to the tkinter.ttk modules. Where, the tkinter package can be seen as an object-oriented layer on top of tcl/tk. It is worth noting that tkinter is already included in Python 3 integrated in the Raspian for Robots operating system.

Some of the generalities of this library are the following:

tcl is a dynamic interpreted programming language like Python.



Tk is a package implemented in C that adds custom commands to create and manipulate GUI widgets.

ttk is a new family of Tk widgets that offers a better experience on different platforms.

Having understood the main function of the Liberator tkinter, in this section some simple examples of how to make interfaces from the Thonny IDE will be made.

The first example is the random number generator. For this, the tkinter and Random libraries must be included (the latter is necessary to be able to generate the numbers).

As can be seen in *code segment 1*, once the libraries are placed in the header there is a function called "update", responsible for generating the number (between 1 and 1000), in addition to publishing it in the interface (line 7).

It is also observed in line 11 the command that repeats the function in a tempo of 500ms, it is thanks to this instruction that the number change can be shown in the interface. If it is not specified, it is not updated on the screen.

It is important to say that in the update function, the line `text2.config` contains different parameters; `text`, which is intended to place a string type character string, `font`, which serves to detail the font type and size that will have the text, `bg`, which specifies the background color and foreground to detail the color of the text font. These parameters are fundamental in the design of an interface, so the user must take each and every one of them into account.

From line 13 to 17 you can find the creation of the main window called `window`, named "Generator", the height and width dimensions, as well as the background color, which in this case is `Orchid4`.

Another important part of the main interface are the Labels, generally used to display messages in a certain space of the main window (lines 19 to 23, 25 and 26) and which also need parameters for their design.

Buttons are also an essential part of the interfaces since they can be used to display customized functions; from lines 28 to 34 there is a button with the design parameters that were used for the case of window and labels, however, 2 new fields can be found; `border` and `command`.

`border` emphasizes the design of the button and `command` points to the "update" function. In short `border` customizes the width of the button border and `command` the function that the program will perform when the button is clicked.

#### Code segment 1. Tkinter window

```

1 import tkinter as tk
2 import random
3
4 def actualizar():
5     num_al=random.randint(1, 1000)
6     texto2.config(text="" + str(num_al),
7                 font=("Arial", 14),
8                 bg="orchid4",
9                 foreground="white")
10    ventana.after(500, actualizar)
11
12 ventana=tk.Tk()
13 ventana.title("Generator")
14 ventana.config(width=250,
15               height=200,
16               bg="orchid4")
17
18 texto1=tk.Label(text="Number: ",
19                font=("Arial", 14),
20                bg="orchid4",

```

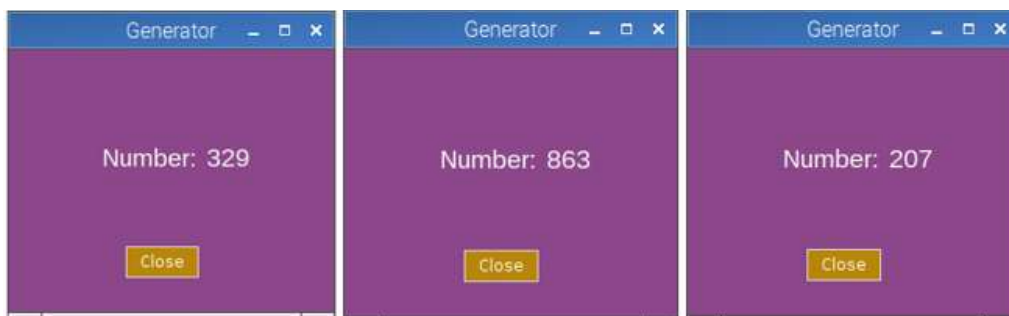
```

21         foreground="white")
22 texto1.place(x=70, y=70)
23
24 texto2=tk.Label(text="0")
25 texto2.place(x=150, y=70)
26
27 Boton=tk.Button(ventana,
28                 text="Close",
28                 command= ventana.destroy,
29                 bg="darkgoldenrod",
30                 foreground="white",
31                 border=0)
32 Boton.place(x=90, y=150)
33
34 ventana.after(500, actualizar)
35 ventana.mainloop()

```

In this way, the result of compiling the program can be seen in *image 5*. In it, 3 windows can be distinguished, which show the different values carried out by the generator. This does not mean that code segment 1 creates 3 windows, but only 1 window, and the values are taken after the time interval specified in the code (500 ms).

**Figurer 5** Final screen (3 transitions)



Having shown a way to display a random value in a small interface in tkinter, the next step is to use the analog channels of the GrovePi+ in conjunction with a graphical window.

For this example, we need to connect two Grove modules; the light sensor and the potentiometer (also known as angle rotary sensor). Where the potentiometer will be connected to the analog port A0 and the light sensor to port A1 of the GrovePi+.

As presented in *code segment 2*, the potentiometer is identified as `potentiometer = 0` and `light_sensor = 1` referencing the GrovePi+ ports. For the proposed program we again make use of the `grovepi` library, in addition to importing all the library modules for the interface.

The `after` method is used again in order to carry out the update for the potentiometer and light sensor values. The functions `def light` and `def pot` are in charge of obtaining the values of each one.

Unlike the first interface that was made in this example we will implement the use of `LabelFrame`, an element provided by tkinter to create custom frames. From line 26 to 44, 2 `LabelFrame`s are coded with their respective geometric editor (grid method), however, it is obvious at first glance that it has not been fully customized by the parameters of each label. This will mark the limits of the frames with the main window of the interface.

Other parameters that can be highlighted from the code are `padx` and `pady`. These elements determine the size of the frame inside the main (root) window.

## Code segment 2. Interface focused on variables

```

1 import tkinter as tk
2 from tkinter import *
3
4 potentiometer=0
5 light_sensor=1
6
7 def luz():
8     val1=grovepi.analogRead(light_sensor)
9     texto3=Label(contenedor1,text="" + str(val1)).place(x=10, y=15)
10    contenedor1.after(500, luz)
11
12 def pot():
13     val2=grovepi.analogRead(potentiometer)
14     texto4=Label(contenedor2, text="" + str(val2)).place(x=10,y=15)
15     contenedor2.after(500, pot)
16
17 root=tk.Tk()
18 root.title("Main Interfaz")
19 root.config(bg= "orchid4")
20
21 contenedor1=LabelFrame(root,
22     text= "Light Sensor",
23     fg= "Blue",
24     padx=65,
25     pady=65)
26 contenedor1.grid(row=0,
27     column=0,
28     padx=15,
29     pady=15)
30 contenedor2=LabelFrame(root,
31     text= "potentiometer",
32     fg= "Blue",
33     padx=65,
34     pady=65)
35 contenedor2.grid(row=0,
36     column=1,
37     padx=15,
38     pady=15)
39
40 texto1=tk.Label(contenedor1,
41     text= "Value: ").pack()
42 texto3=tk.Label(contenedor1,
43     text= "0").place(x=10, y=15)
44 texto2=tk.Label(contenedor2,
45     text= "Value: ").pack()
46 texto4=tk.Label(contenedor2,
47     text= "0").place(x=10, y=15)
48
49 root.after(500, luz)
50 root.after(500, pot)
51 root.mainloop()

```

## Matplotlib

It is a library that helps the user to plot data in figures (or Figure), it contains one or more Axes (Area where the points can be specified in terms of XY or XYZ coordinates in case of 3D graphics) and a variety of methods that can give dynamism to the graphics. The easiest way to create a figure with axes is using `pyplot.subplots` where it is important to note that in order to display the figure you must call the `plt.show()` method.

The above interfaces can be complemented with graphics by using the matplotlib library. However, the Raspbian for Robots operating system lacks dependencies that slow down the installation of the library through pip, that is why we choose to use an alternative method as shown in *command 7*.

This command must be run from the terminal, if you try to install from the Thonny package manager is very likely not allowed and only manages to remove dependencies linked to the grovepi library.

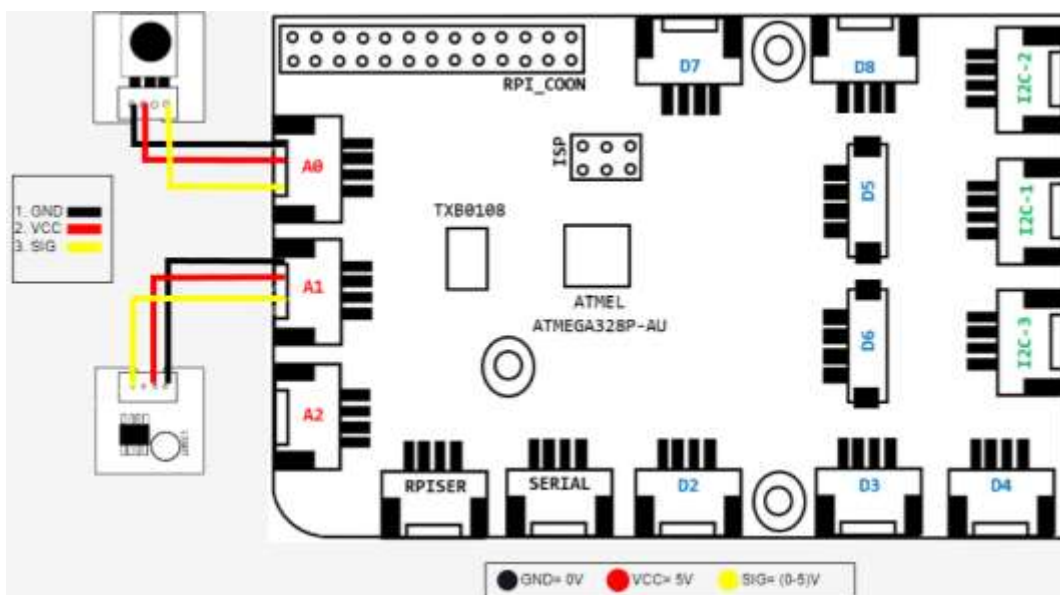
```
sudo apt install python3-matplotlib
```

Command 7 Matplotlib installation

For this circuit you need the potentiometer and the light sensor again, the potentiometer must be connected to channel A0 and the light sensor to channel A1 of GrovePi+ as shown in *image 6*. Remembering that GrovePi+ contains 3 analog ports you can change the connection order of the modules (if required), keeping in mind that you must additionally change the configuration in *code segment 3*.

From figure 6, as it is the first connection diagram seen in this work, it is necessary to mention that the color code was established according to the Grove Cable.

**Figure 6** Connection diagram



Once the compatible library has been downloaded, the program is presented in code segment 4. It has the function of plotting the readings of a potentiometer and the light sensor (grove rotary angle sensor & grove light sensor).

In the code you can see the different sections that conform it: functions in charge of providing the sensor readings and functions that draw the lines of the graph according to the value obtained from the code blocks `read_pot` and `luminosidad`.

The way in which the graphs will be presented will be by separate windows, in this case we do not use Tkinter, only Matplotlib. It should be noted that the program is comprised of 3 instructions that, enclosed in the while loop, carry out the plotting of the 2D lines in a continuous manner.

Going deeper into the functions `grafico(frame)` and `grafico1(frame)`, the `append` method is used as the main element in the assignment of values to the `x_data`, `y_data`, `x1_data` and `y1_data` arrays, which makes efficient data processing possible. These functions are a fundamental part of the trafication process since lines 56 and 57 contain the instructions that make possible the update of values; `FuncAnimation(figure, graph, interval=400)` and `FuncAnimation(figure1, graph1, interval=400)` besides containing the same update interval (400 ms) point to their respective space or "plot" and to the function of the variable they represent (`grafico` focuses on the potentiometer and `grafico1` on the light sensor).

## Code segment 3 Creation of graphics

```

1 from tkinter import *import time
2 import grovepi
3 import matplotlib.pyplot as plt
4 from datetime import datetime
5 from matplotlib import pyplot
6 from matplotlib.animation import FuncAnimation
7
8 plt.style.use('ggplot')
9 x_data=[]
10 y_data=[]
11 x1_data=[]
12 y1_data=[]
13
14 figure= pyplot.figure()
15 line, =pyplot.plot_date(x_data, y_data, '-')
16
17 figure1= pyplot.figure()
18 line1, =pyplot.plot_date(x_data, y1_data, '-')
19
20 potentiometer = 0
21 light_sensor = 1
22
23 grovepi.pinMode(potentiometer,"INPUT")
24 grovepi.pinMode(light_sensor,"INPUT")
25
26 time.sleep(5)
27
28 def grafico(frame):
29     x_data.append(datetime.now())
30     pase=lectura_pot()
31     y_data.append(pase)
32     line.set_data(x_data, y_data)
33     figure.gca().relim()
34     figure.gca().autoscale_view()
35     return line,
36
37 def grafico1(frame):
38     x1_data.append(datetime.now())
39     sensorluminoso=luminosidad()
40     y1_data.append(sensorluminoso)
41     line1.set_data(x1_data, y1_data)
42     figure1.gca().relim()
43     figure1.gca().autoscale_view()
44     return line1,
45
46 def lectura_pot():
47     sensor_value = grovepi.analogRead(potentiometer)
48     return(sensor_value)
49
50 def luminosidad():
51     sensor_value1 = grovepi.analogRead(light_sensor)
52     return (sensor_value1)
53
54 while True:
55     try:
56         animation2= FuncAnimation(figure, grafico, interval=400)
57         animation= FuncAnimation(figure1, grafico1, interval=400)
58         pyplot.show()
59     except IOError:
60         print ("Error")

```

## XlsxWriter

XlsxWriter is an open source Python API for writing files in the Excel 2007+ XLSX file format. With the API, you can write text, formulas, numbers and hyperlinks to multiple worksheets. In addition, the API allows you to insert charts, merge cells, format cells, apply filters, validate data, insert PNG/JPEG/BMP/WMF/EMF Figures, use multi-format strings and more.

XlsxWriter provides more Excel functions than any of the alternative Python modules. In addition, it provides a high accuracy rate when creating new Excel files; in most cases, files produced with XlsxWriter are 100% equivalent to files produced by Excel.

From this description, in this section the "Recording" of data implementing an analog sensor will be performed.

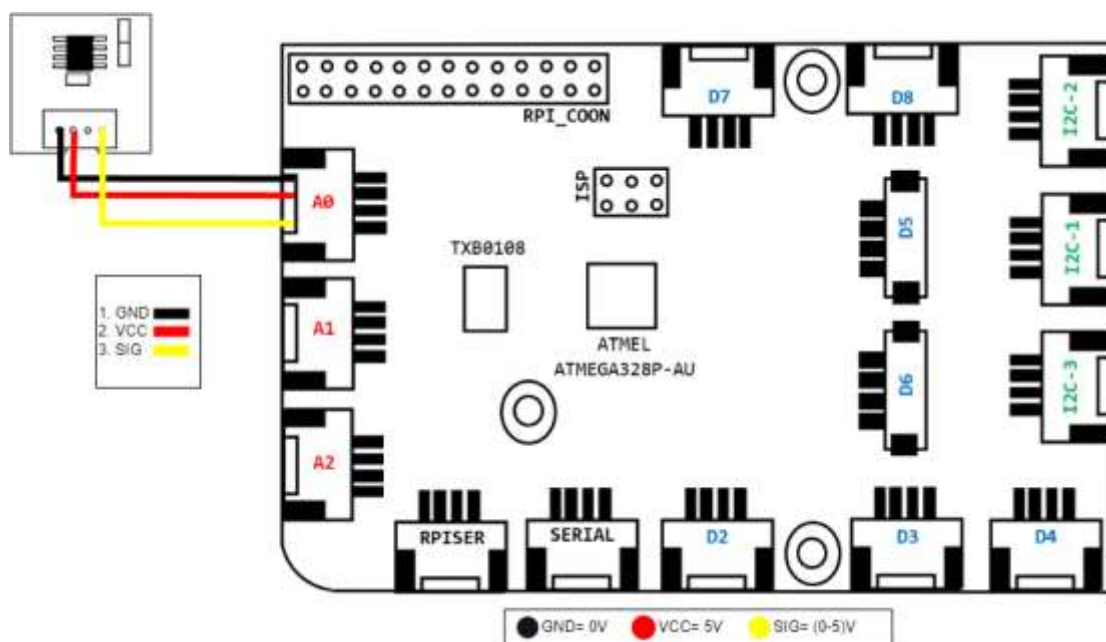
Initially it is necessary to install the library, for this we use pip, with the *command 8* executed from the terminal.

```
pip install XlsxWriter
```

*Command 8 Xlsxwriter installation*

The sensor to be used will be the temperature sensor as shown in *Figure 7*.

**Figure 7** Connection diagram



Once the connection has been made, the program for this activity is the one seen in *code segment 4*. It is in this code that a class is presented that has the objective of obtaining values from various sensors, thus providing an alternative to the usual `grovepi.analogRead()` instruction provided by the `grovepi` library. Highlighting that it favors the compilation of the program in terms of time.

Some of the lines that are indispensable in the elaboration of the script are described:

The XlsxWriter library requires specific syntax for file creation, saving and saving. Line 11 describes how to create the file and gives the option to name it. Line 12 adds spreadsheets to the file.

Line 15 contains an instruction which involves the execution of *code segment 5* yielding the temperature value.

To store data the instruction `hoja.write` can be displayed in lines 18 and 19 and as arguments the location (row, column, data to be stored).

In order to save the file, line 25 contains the instruction `archivo.close()`.

The for loop performs the storage of the temperature value 50 times, therefore, the final file will contain 2 rows; one for the timestamp and one for the temperature value.

#### Code segment 4. Data storage with Xlsxwriter

```

1 import grovepi
2 import dht11_modulo
3 import time
4 import xlsxwriter
5
6 i=0
7 h=0
8 temperatura=0
9 grovepi.pinMode(temperatura,"INPUT")
10
11 archivo=xlsxwriter.Workbook('GrabadoDeLecturas.xlsx')
12 hoja=archivo.add_worksheet()
13
14 for c in range (0,50):
15     val2=dht11_modulo.lectura_pot(temperatura)
16     hora=time.strftime("%X")
17     hoja.write(i,1,val2)
18     hoja.write(h,0,hora)
19     print(val2)
20     i=i+1
21     h=h+1
22     time.sleep(3)
23
24 archivo.close()

```

Another thing that can be added from the class of *code segment 5* is the versatility of adding or changing sensors in this section, eliminating a possible problem (if it is the case of changing the type of sensor) of having to resort to the main program and look for the lines to change it.

For this it is more advisable to have a class intended only for sensors.

#### Code segment 5 Class that obtains sensor readings

```

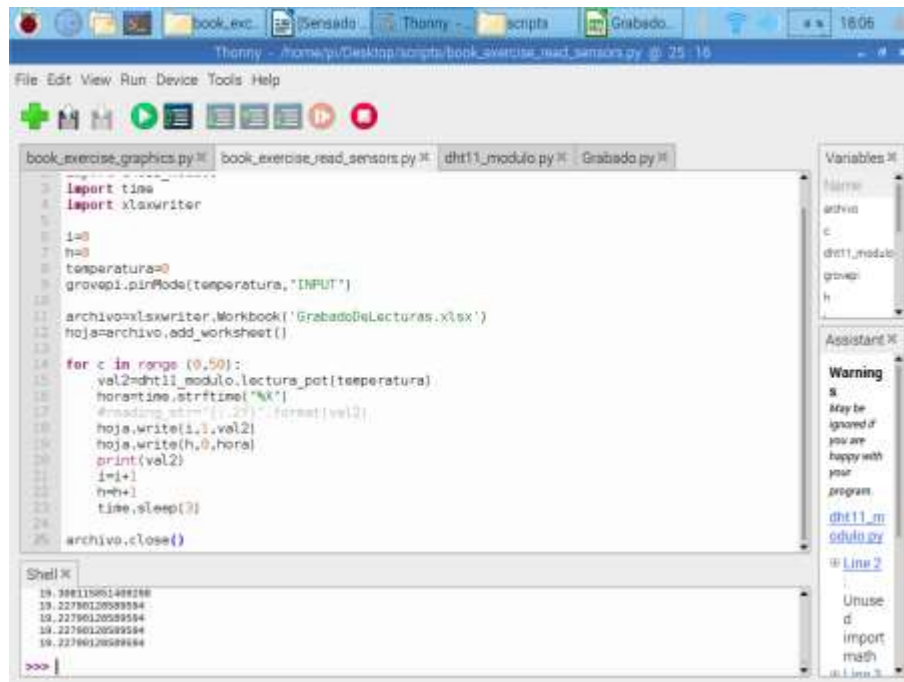
1 import grovepi
2 import math
3 import time
4
5 def lectura_hummodulo():
6     sensor_value2= grovepi.analogRead(moisture_sensor)
7     return (sensor_value2)
8
9 def lectura_lummodulo():
10    sensor_value1= grovepi.analogRead(light_sensor)
11    return (sensor_value1)
12
13 def lectura_pot():
14    sensor_value= grovepi.analogRead(temperatura)
15    return (sensor_value)

```

Figure 8 shows a first execution of the code in which the temperature values can be observed in the console, in order to check the data with those of the final file. Since the main program only takes 50 values, it provides a way to check the veracity of the data by comparing the data printed in the Thonny console with those saved in the Excel file.



Figure 8 Console result



```

1 import time
2 import xlswriter
3
4 i=0
5 h=0
6 temperatura=0
7 grovapi.pirNode(temperatura,"INPUT")
8
9 archivo=xlswriter.Workbook('GrabadoDeLecturas.xlsx')
10 hoja=archivo.add_worksheet()
11
12
13
14 for c in range(0,50):
15     val2=dht11_modulo.lectura_pot(temperatura)
16     hora=time.strftime("%X")
17     #reading_str="{:1.2f} - {}".format(val2)
18     hoja.write(i,1,val2)
19     hoja.write(h,0,hora)
20     print(val2)
21     i=i+1
22     h=h+1
23     time.sleep(2)
24
25 archivo.close()

```

Shell

```

19.388115951489188
19.22796120589554
19.22796120589554
19.22796120589554
19.22796120589554

```

Variables

Warning  
May be ignored if you are happy with your program

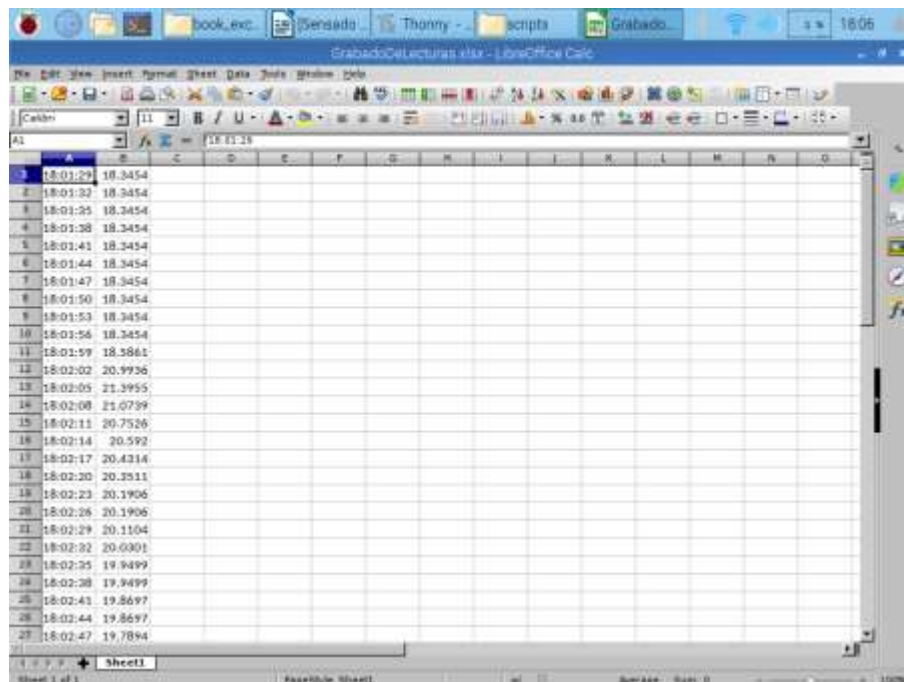
dht11\_m  
odule.py

Unuse  
d  
import  
math

Now it is time to open the generated file, in order to find it just open the folder where the main program is hosted. Reviewing the file you can find the data of the Figure 9.

The type of data with which each reading was saved are 2; time and string. This feature must be taken into account because, if you plan to use this data to insert graphs or perform calculations, the string data belonging to the temperature (Column B) will have to be converted into an integer or floating data type. In this way it is possible to develop a tool very similar to Data Streamer (Excel add-in whose function is to capture data acquired by the Arduino in serial form).

Figure 9 View to final file



Time	Temperature
18:01:29	18.3454
18:01:32	18.3454
18:01:35	18.3454
18:01:38	18.3454
18:01:41	18.3454
18:01:44	18.3454
18:01:47	18.3454
18:01:50	18.3454
18:01:53	18.3454
18:01:56	18.3454
18:01:59	18.5861
18:02:02	20.9936
18:02:05	21.3955
18:02:08	21.0739
18:02:11	20.7526
18:02:14	20.592
18:02:17	20.4316
18:02:20	20.3911
18:02:23	20.1906
18:02:26	20.1906
18:02:29	20.1104
18:02:32	20.0901
18:02:35	19.9499
18:02:38	19.9499
18:02:41	19.8697
18:02:44	19.8697
18:02:47	19.7894

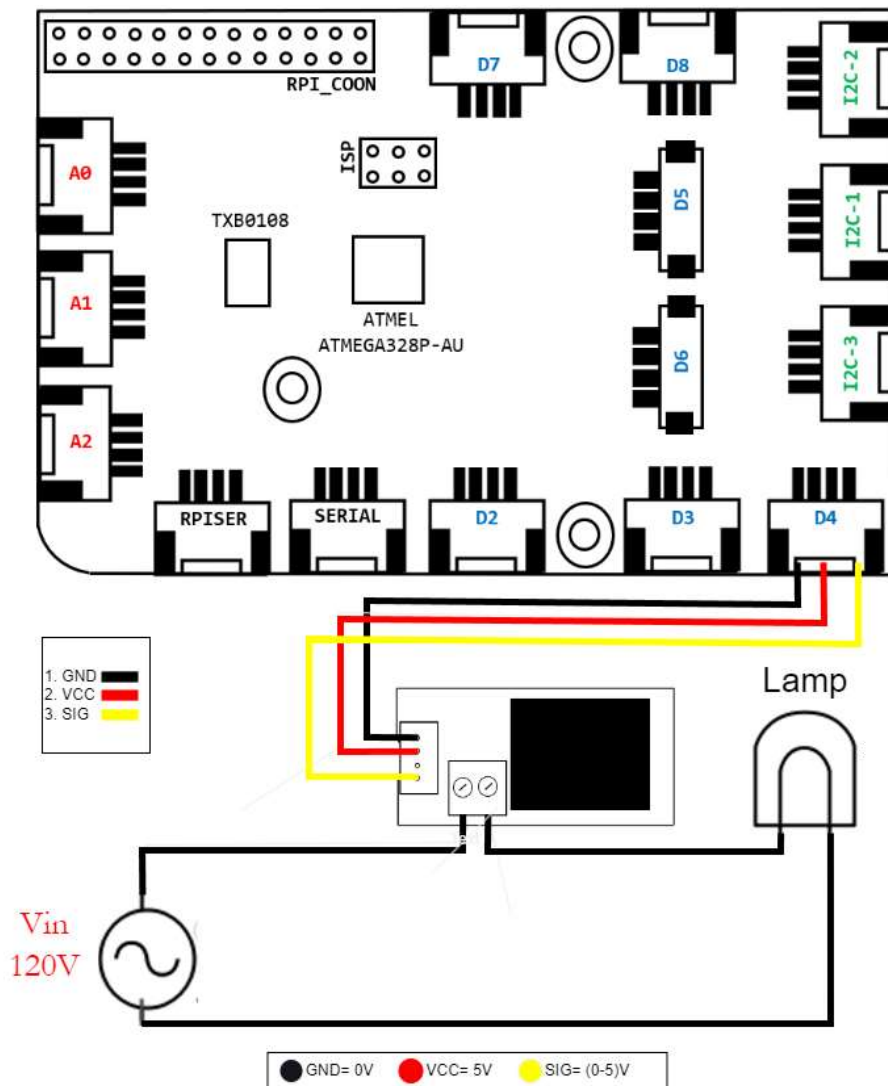
## Circuits with actuators and other components

### Relay

This is a basic circuit whose objective is to demonstrate the use of the relay. It is important to emphasize that in case of not having the Grove module, a generic relay can replace it without any problem, although it is necessary to have a Grove Cable for a correct communication with the GrovePi+ hat.

As shown in Figure 10, the relay must be connected to port D4 of GrovePi+ and the bulb must be open in one of its two connections.

**Figure 10** Connection diagram



In code segment 6 you can see how the program only outputs values of 0 and 1 (0v and 5v) for 2 second intervals. 2 seconds the relay will let the light bulb turn on and 2 seconds will turn it off in an infinite loop thanks to the while loop.

Despite being a relatively short code, it has the virtue of being very understandable as if it were an Arduino. It can be noticed that the number of libraries to use are much less than in previous circuits. And it could even be very easily related to a switch on and off an LED. The instruction that allows this state switching is located in line 10 and 14 through `grovepi.digitalWrite()`.

## Code segment 6 Use of the relay module

```

1 import time
2 import grovepi
3
4 relay = 4
5
6 grovepi.pinMode(relay,"OUTPUT")
7
8 while True:
9     try:
10        grovepi.digitalWrite(relay,1)
11        print ("on")
12        time.sleep(2)
13
14        grovepi.digitalWrite(relay,0)
15        print ("off")
16        time.sleep(2)
17
18    except KeyboardInterrupt:
19        grovepi.digitalWrite(relay,0)
20        break
21    except IOError:
22        print ("Error")

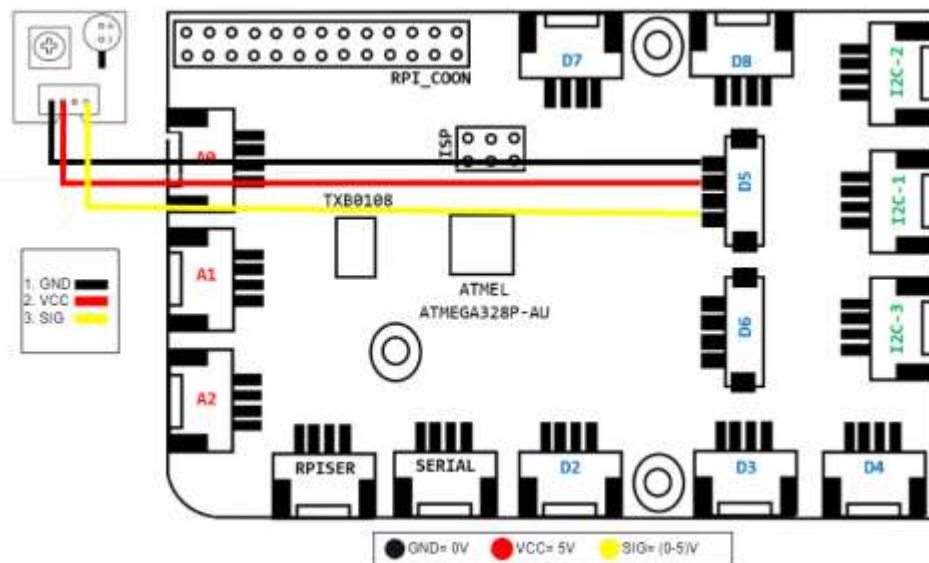
```

## PWM

The PWM pulse has several uses, from speed control of a motor to a variable frequency drive and even being part of a digital PID controller. That is why in this section we present a simple code that demonstrates its operation using the Grove led pack.

The connection of this circuit is limited to use a single module, being a PWM it is important to consider using a digital port of the GrovePi+ with the ability to throw a pulse of this type. As can be seen in Figure 11, port D5 is the one chosen.

**Figure 11** Connection diagram



As for the program, code segment 7 provides a quick way to display the PWM by means of 2 for loops. With them it will be possible to observe in the physical assembly how the LED changes the brightness level continuously up and down thanks to the described while loop.

It should be remembered that a PWM pulse is between values from 0 to 255, although the changes are visible in shorter values and with longer delays, that is why the voltage will only be varying with values between 0 and 20.

## Code segment 7 PWM operation

```

1 import time
2 from grovepi import *
3
4 led = 5
5 z=0
6 pinMode(led,"OUTPUT")
7
8 while True:
9   for i in range(0,20):
10    z+=1
11    analogWrite(led,z)
12    print(z)
13    time.sleep(0.1)
14   for j in range(0,20):
15    z-=1
16    analogWrite(led,z)
17    print(z)
18    time.sleep(0.1)

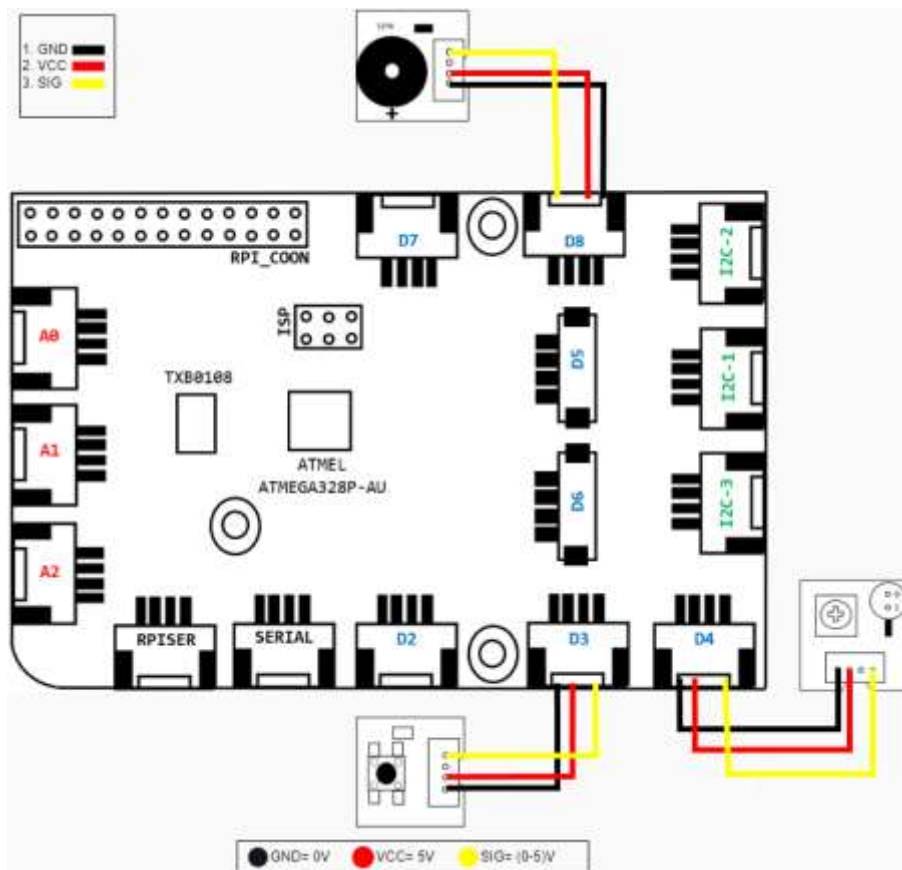
```

## Buzzer

For this circuit we will make use of 3 components, the buzzer module, a button and the led pack. Each time the button is pressed, the buzzer will have to emit sound and at the same time the led will have to be in high state.

In the diagram of Figure 12 it can be seen how only digital channels are used due to the nature of the modules to be used. The buzzer must be connected to channel D8, the led to D4 and the button to D3.

**Figure 12** Connection diagram



In the code segment 8 above you can see how the while loop encompasses the operation of the circuit by means of a conditional if-else. The if key compares if the input status coming from the button is high (button pressed), if it is true both the buzzer and the led will be activated.

## Code segment 8. Control of the buzzer module

```

1 import time
2 import grovepi
3
4 buzzer = 8
5 button = 3
6 led = 4
7
8
9 grovepi.pinMode(button,"INPUT")
10 grovepi.pinMode(buzzer,"OUTPUT")
11 grovepi.pinMode(led,"OUTPUT")
12
13 while True:
14     try:
15         if(grovepi.digitalRead(button)==1):
16             grovepi.digitalWrite(buzzer,1)
17             grovepi.digitalWrite(led,1)
18         else:
19             grovepi.digitalWrite(buzzer,0)
20             grovepi.digitalWrite(led,0)
21
22     except IOError:
23         print ("Error")

```

## CD motor control

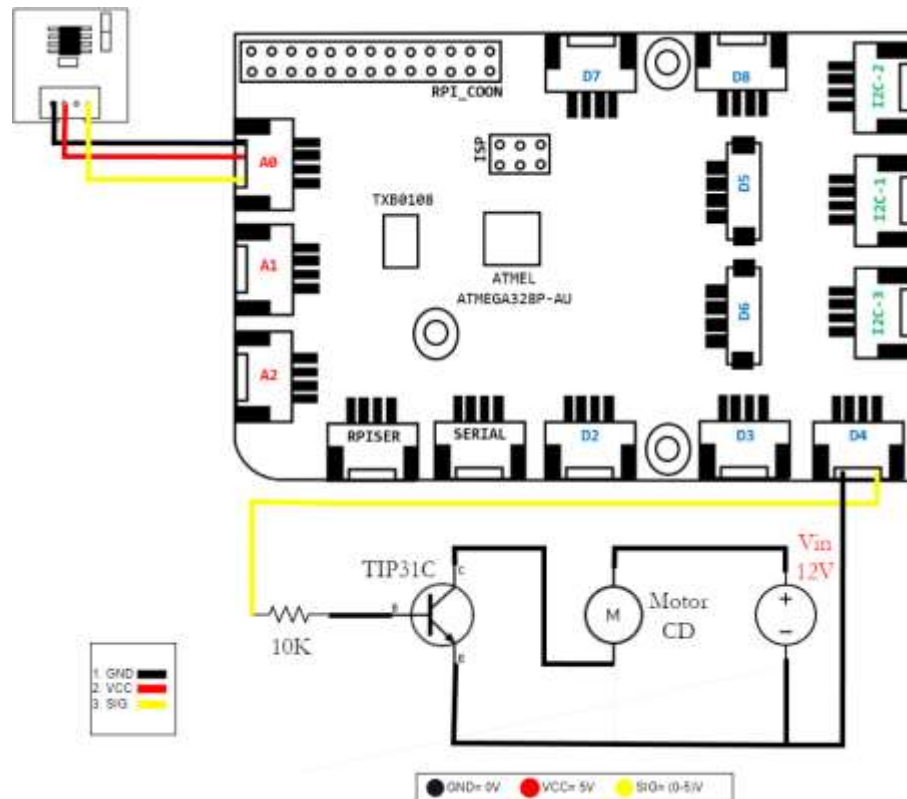
For this circuit another electrical component is presented, the transistor. Since the motor to be used is one of 12V and is of direct current, it is required to compensate the consumption current, that is why the tip31C will be used taking into account that it is a BJT in Darlington arrangement.

As can be seen in *Figure 13*, the circuit requires the following components;

- GrovePi+
- Protoboard.
- Grove Temperature Sensor v1.2
- Resistor 10k
- Tip31C
- Motor 12v DC
- 12v charger / Voltage source.

The ports to be used are A0 and D4.

Figure 13 Connection diagram



According to code segment 9, the motor will be triggered every time the temperature sensor reaches values above 20° C in addition to printing the current value being detected on the console. A while loop is used in this case for constant temperature monitoring.

#### Code segment 9 DC motor control

```

1 import time
2 import grovepi
3 from grovepi import *
4
5 led = 4
6 temperature=0
7
8 pinMode(led,"OUTPUT")
9 pinMode(temperature,"INPUT")
10
11 while True:
12     val=grovepi.temp(temperature,'1.2')
13     print(val)
14     if(grovepi.temp(temperature,'1.2')>20):
15         analogWrite(led,255)
16     else:
17         analogWrite(led,0)

```

#### Grove moisture sensor

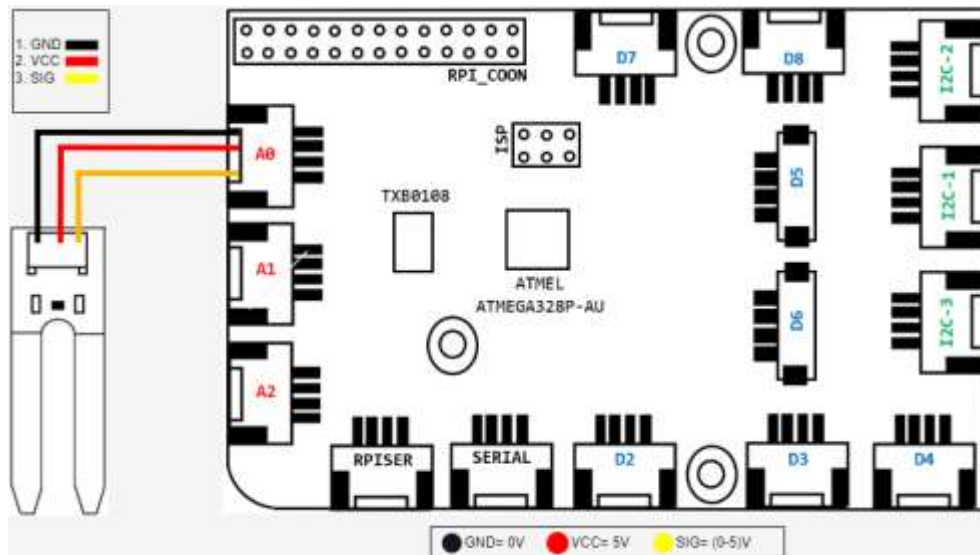
This is a circuit widely used as far as GrovePi+ is concerned, since only the soil moisture sensor and the base program provided by Dexter Industries are needed.

Since it is an analog sensor, it can be used in any of the 3 ports provided by the GrovePi+, in this case the A0 channel will be used as shown in Figure 14.

One of the things to keep in mind about the humidity sensor is that when placing it in the plant, care must be taken not to completely submerge the module, respecting the division between the electrodes and the connection port.



Figure 14 Connection diagram



Returning to the interfaces, this monitoring will be complemented with a small window in which the value recorded by the humidity sensor can be observed (*code segment 10*). So at this point the structure of the program can be repetitive since the bases of the potentiometer and light sensor interface are taken. For this interface only 1 single label (line 19) will be taken as a reference.

In *code segment 10* there is also a block that involves the use of Figures. Being the first time that these design elements are used in this work, it is important to keep in mind that both the format of the illustrations and their dimensions must be respected in order not to generate irregularities in the final window. As for the format, the ideal format to work with is the jpg, while the dimensions may vary depending on the measures assigned to the main window (root), if the Figure is larger in resolution compared to the main interface, the latter will be covered by the Figure to be placed on top of any element (buttons, frames or labels). If the format is not the right one, compilation errors will occur or it will simply not be displayed in the window. In the code, lines 36, 37 and 38 load an illustration that will serve as an icon to give a better presentation to the window. The Figure is instantiated with the variable "img" and positioned with the function place().

#### Code Segment 10 Moisture monitoring

```

1 import tkinter as tk
2 from tkinter import *
3 import grovepi
4
5 hum=0
6
7 grovepi.pinMode(hum,"INPUT")
8
9 def humedad():
10     val1 = grovepi.analogRead(hum)
11     texto3=Label(contenedor1,
12     text="" + str(val1)).place(x=10,y=15)
13     contenedor1.after(500, humedad)
14
15 root=tk.Tk()
16 root.title("Plant")
17 root.config(bg="orchid4")
18
19 contenedor1=LabelFrame(root,
20     text="Soil Moisture",
21     fg="blue",
22     padx=65,
23     pady=65.)
24 contenedor1.grid(row=0,
25     column=0,
26     padx=15,
27     pady=15)

```



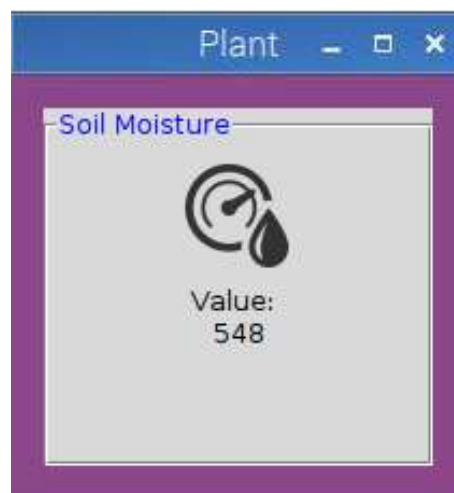
```

28
29 texto1=tk.Label(contenedor1,
30     text="Value: ").pack()
31
32 texto3=tk.Label(contenedor1,
33     text="0").place(x=10,y=15)
34
35 img=tk.PhotoFigure(file="hum.png")
36 lbimg=tk.Label(root, Figure=img)
37 lbimg.place(x=80,y=40)
38
39 root.after(500, humedad)
40 root.mainloop()

```

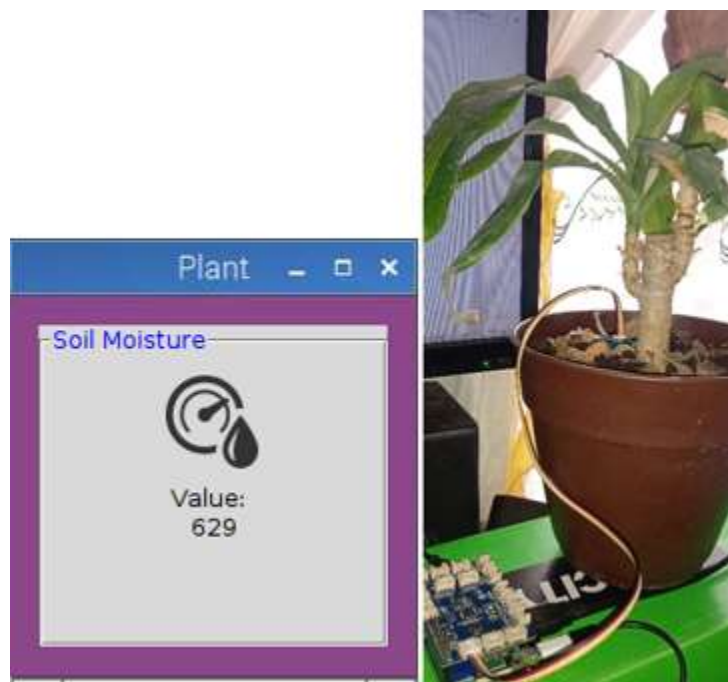
It is important to mention that the value presented in Figure 15 is for an unwatered plant. In this way we have a reference value with which to make a comparison later on.

**Figure 15** Window for plant monitoring



When the plant is irrigated, in Figure 16 the value changes to 626 obtaining a very small change for the range of values that the sensor comprises (0 to 1024). This circuit has possibilities to be complemented with a DC motor that has the function of a water pump with the help of the relay, thus implementing the automatic control depending on the state of the plant

**Figure 16** Humidity level after adding water



## Acknowledgments

This paper would not have been possible without the support of the Universidad de Ixtlahuaca CUI and the TecNM campus Jocotitlán, who mainly provided the material seen during this work.

## Conclusions

During the present work the use of different Grove modules was approached, with this it was possible to implement both the control and the visualization through simple graphical interfaces taking advantage of the benefits offered by Python. Implementing analog sensors with the graphical windows represents an alternative for similar desktop applications, demonstrating that the graphics have a relatively fast update time, the ease of programming the control of the components and the wide variety of libraries that Python has, allowing the addition of more features without aggressively consuming Raspberry Pi resources.

In terms of programming, the grovepi library is of great help to the user by providing understandable instruction lines that allow manipulating different actuators as can be seen in the motor control circuit on CD. In conclusion, the use of GrovePi+ is a suitable alternative to make visualization and control interfaces that depending on the robustness of the system needed by the user, the use of the different ports and Python will be less effort compared to other hats that work with C programming.

## References

- ¿Qué es Python? - Explicación del lenguaje Python - AWS. (n.d.). Amazon Web Services, Inc. <https://aws.amazon.com/es/what-is/python/>
- Anicai, C. y Shakir, MZ (marzo de 2023). IoT and Machine Learning Enabled Estimation of Health Indicators from Ambient Data. IEEE. <https://doi.org/10.1109/WCNC55385.2023.10119030> En 2023, IEEE Wireless Communications and Networking Conference (WCNC) (págs. 1-6). IEEE.
- Balasubramaniyan, C., & Manivannan, D. (2016). IoT Enabled Air Quality Monitoring System (AQMS) using Raspberry Pi. Indian Journal of Science and Technology, 9(39). <https://doi.org/10.17485/ijst/2016/v9i39/90414>
- Bell, C. (2021). *Beginning IoT projects: Breadboard-less Electronic Projects*. Apress.
- Creating, viewing, and saving Matplotlib Figures — Matplotlib 3.7.2 documentation. (n.d.). <https://matplotlib.org/stable/users/explain/figures.html#figure-explanation>
- D BÜCH, D., B., & M ESCH, M., E. (2023). A Testbed for Smart City Applications and Architectures. IEEE. <https://doi.org/10.1109/COINS57856.2023.10189229>. In 2023, International Conference on Omni-layer Intelligent Systems (COINS) (págs. 1-6). IEEE.
- Getting Started with Ubuntu Core for Raspberry Pi 3. (n.d.). (n.p.): PE
- Gonzalez, R. (2021, December 15). Guía para manejo de tarjetas programables con Python y JavaScript para Internet de las Cosas. URI: <http://repositorio.uts.edu.co:8080/xmlui/handle/123456789/8235>
- GrovePi Port description. (2017, March 21). Dexter Industries. <https://www.dexterindustries.com/GrovePi/engineering/port-description/>
- Ltd, A. P. (2023). API Python de código abierto para hojas de cálculo de Google crear y compartir hojas de cálculo. Aspose Pty. Ltd. <https://products.fileformat.com/es/spreadsheet/python/xlsxwriter/#:~:text=XlsxWriter%20es%20una%20API%20de,en%20varias%20hojas%20de%20trabajo.>
- Mg, I. Y. M. L. (2019, August 1). Determinación del caudal Ecológico en la Microcuenca del Río Cutuchi mediante el método de Tennant en Python, periodo 1988 – 2014. <http://repositorio.utc.edu.ec/handle/27000/6107>

Paguayo. (2022). ¿Que es Raspberry Pi? - Raspberry Pi. Raspberry Pi. <https://raspberrypi.cl/que-es-raspberry/>

Pawar, S., Kelkar, S., Khire, N., Khairnar, T. y Kharabe, M. (marzo de 2023). AQI Monitoring and Predicting System. IEEE. <https://doi.org/10.1109/ESCI56872.2023.10099645>. In 2023, IEEE International Conference on Emerging Smart Computing and Informatics (ESCI) (págs. 1-6). IEEE.

Press. Molloy, D. (2019). Raspberry Pi® a fondo para desarrolladores. España: Marcombo.

tkinter — Python interface to Tcl/Tk. (n.d.). Python Documentation. <https://docs.python.org/3/library/tkinter.html#architecture>

Vila, M., Sancho, M. R., & Teniente, E. (2023, March). Monitoring, IoT Devices, and Semantics. IEEE. <https://doi.org/10.1109/PerComWorkshops56833.2023.10150279>. In 2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) (pp. 309-312). IEEE.

Wright, H. C., Cameron, D. D., & Ryan, A. J. (2022). FoamPi: an open-source Raspberry Pi based apparatus for monitoring polyurethane foam reactions. *HardwareX*, 12, e00365. <https://doi.org/10.1016/j.ohx.2022.e00365>