

INTERFACING DHT11 TEMPERATURE SENSOR WITH THE RASPBERRY PI 5

Detail Contents

USING THE DHT11 TEMPERATURE SENSOR WITH THE RASPBERRY PI	1
1 SETTING UP ENVIRONMENT FOR PYTHON CODE	3
1.1 Installing dht11 library	5
1.2 Installing Rpi.lgpio library	6
1.3 Verify your installations	8
2 PYTHON CODE FOR DHT11 MODULE	11
3 RUN THE PROGRAM	16
4 IMPROVING DHT11 LIBRARY	19
4.1 Get dht11 into your project directory	19
4.2 Edit dht11 program.	20
4.3 Edit dht11 library	21
5 ADDITIONAL INFO – MAKING EXECUTABLE APP IN RASPBERRY PI	23

1 SETTING UP ENVIRONMENT FOR PYTHON CODE

This section will explain the programming procedures of Raspberry Pi in two different environments.

You can choose any one of these methods:

Method 1: Virtual environment

Method 2: System-wide environment

Now, we need to prepare the Raspberry Pi by installing necessary libraries. This tutorial is for Debian 12 Bookworm Operating System.

Two libraries that will be installed are:

1. Rpi.lgpio library by Dave Jones (<https://rpi-lgpio.readthedocs.io/en/latest/>)
2. Dht11 library by Zoltan Szarvas (https://github.com/szazo/DHT11_Python)

What is a virtual environment?

Virtual environment is an isolated environment for running python code. It allows you to manage dependencies for different projects separately, ensuring that libraries used for one project do not interfere with others.

When you create a virtual environment, a folder directory is created. When you install libraries inside a virtual environment, the libraries' folders will be kept in the virtual environment folder, isolated from the global Python installation. This prevents version conflicts between different projects and maintains a clean system environment.

1.1 Installing dht11 library

i) Method 1: Dht11 library installation in virtual environment

- 1) Create a virtual environment. In this case, we named the virtual environment as “myenv”.

```
controleasy@raspberrypi:~ $ python3 -m venv myenv
```

- 2) Activate the virtual environment.

```
controleasy@raspberrypi:~ $ source myenv/bin/activate
```

- 3) Install dht11 library:

```
(myenv) controleasy@raspberrypi:~ $ pip install dht11
```

Info: to exit the virtual environment, you can simply write “deactivate” into the terminal:

```
controleasy@raspberrypi:~ $ source myenv/bin/activate  
(myenv) controleasy@raspberrypi:~ $ deactivate  
controleasy@raspberrypi:~ $
```

ii) Method 2: Dht11 library installation system-wide

- 1) Clone the dht11 repository from GitHub into your Raspberry Pi, and then navigate to the cloned repository's directory.

```
controleasy@raspberrypi:~ $ git clone https://github.com/szazo/DHT11_Python.git  
Cloning into 'DHT11_Python'...  
remote: Enumerating objects: 73, done.  
remote: Total 73 (delta 0), reused 0 (delta 0), pack-reused 73 (from 1)  
Receiving objects: 100% (73/73), 16.85 KiB | 1.30 MiB/s, done.  
Resolving deltas: 100% (26/26), done.  
controleasy@raspberrypi:~ $ cd DHT11_Python  
controleasy@raspberrypi:~/DHT11_Python $
```

- 2) Install the dht11 library by running setup.py located in the cloned repository

```
controleasy@raspberrypi:~/DHT11_Python $ python3 setup.py install --user
```

1.2 Installing Rpi.lgpio library

We will be using the new **rpi.lgpio** library because this gpio library is compatible with Raspberry Pi 3, Raspberry Pi 4, and Raspberry Pi 5.

Warning:

You cannot install rpi-lgpio and rpi-gpio (aka RPi.GPIO, the library it emulates) at the same time, in the same Python environment. Both packages attempt to install a module named RPi.GPIO and obviously this will not work.

Note: when you install dht11 library, rpi.gpio library will also be installed automatically as its dependencies. That's why the rpi.lgpio library installation process is done after the dht11 library installation.

i) Method 1: Rpi.lgpio installation in virtual environment

- 1) Activate a virtual environment. In this line, we activated the virtual environment "myenv" that was created previously.

```
controleasy@raspberrypi:~ $ source myenv/bin/activate
```

- 2) Remove rpi.gpio library

```
(myenv) controleasy@raspberrypi:~ $ pip uninstall rpi-gpio
Found existing installation: RPi.GPIO 0.7.1
Uninstalling RPi.GPIO-0.7.1:
  Would remove:
    /home/controleasy/myenv/lib/python3.11/site-packages/RPi.GPIO-0.7.1.egg-info
    /home/controleasy/myenv/lib/python3.11/site-packages/RPi/*
Proceed (Y/n)? y
Successfully uninstalled RPi.GPIO-0.7.1
```

- 3) Install rpi.lgpio library

```
(myenv) controleasy@raspberrypi:~ $ pip install rpi-lgpio
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting rpi-lgpio
  Downloading https://www.piwheels.org/simple/rpi-lgpio/rpi_lgpio-0.6-py3-none-any
  hl (11 kB)
Collecting lgpio>=0.1.0.1
  Downloading lgpio-0.2.2.0-cp311-cp311-manylinux_2_34_aarch64.whl (364 kB)
  364.8/364.8 kB 2.9 MB/s eta 0:00:00
Installing collected packages: lgpio, rpi-lgpio
Successfully installed lgpio-0.2.2.0 rpi-lgpio-0.6
```

ii) **Method 2: Rpi.lgpio installation system-wide environment**

- 1) Update the local package index with the latest available versions of packages. Then, upgrade installed packages of your Raspberry Pi to their latest versions:

```
controleasy@raspberrypi:~ $ sudo apt update
```

```
controleasy@raspberrypi:~ $ sudo apt upgrade
```

- 2) Remove rpi.gpio library

```
controleasy@raspberrypi:~ $ sudo apt remove python3-rpi.gpio
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libcamera0.2 libraspberrypi0 libwpe-1.0-1 libwpebackend-fdo-1.0-1 rpi.gpio-common
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  python3-rpi.gpio
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 96.3 kB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 243830 files and directories currently installed.)
Removing python3-rpi.gpio (0.7.1~a4-1+b4) ...
```

- 3) Install rpi.lgpio library

```
controleasy@raspberrypi:~ $ sudo apt install python3-rpi-lgpio
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libcamera0.2 libraspberrypi0 libwpe-1.0-1 libwpebackend-fdo-1.0-1 rpi.gpio-common
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  python3-rpi-lgpio
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 12.3 kB of archives.
After this operation, 56.3 kB of additional disk space will be used.
Get:1 http://archive.raspberrypi.com/debian bookworm/main arm64 python3-rpi-lgpio all 0.4-1~rpt1 [12.3 kB]
Fetched 12.3 kB in 1s (14.9 kB/s)
Selecting previously unselected package python3-rpi-lgpio.
(Reading database ... 243820 files and directories currently installed.)
Preparing to unpack .../python3-rpi-lgpio_0.4-1~rpt1_all.deb ...
Unpacking python3-rpi-lgpio (0.4-1~rpt1) ...
Setting up python3-rpi-lgpio (0.4-1~rpt1) ...
```

1.3 Verify your installations

In any part of these tutorial, you can check if a library is installed correctly by writing the following command to display the list of libraries installed in any environment, virtual or system-wide:

```
~ $ pip list
```

Example:

The following are examples of “pip list” output for “myenv” virtual environment.

- a) After just creating the virtual environment

```
(myenv) controleasy@raspberrypi:~ $ pip list
Package      Version
-----
pip          23.0.1
setuptools   66.1.1
```

- b) After installing dht11 library

```
(myenv) controleasy@raspberrypi:~ $ pip list
Package      Version
-----
dht11        0.1.0
pip          23.0.1
RPi.GPIO     0.7.1
setuptools   66.1.1
```

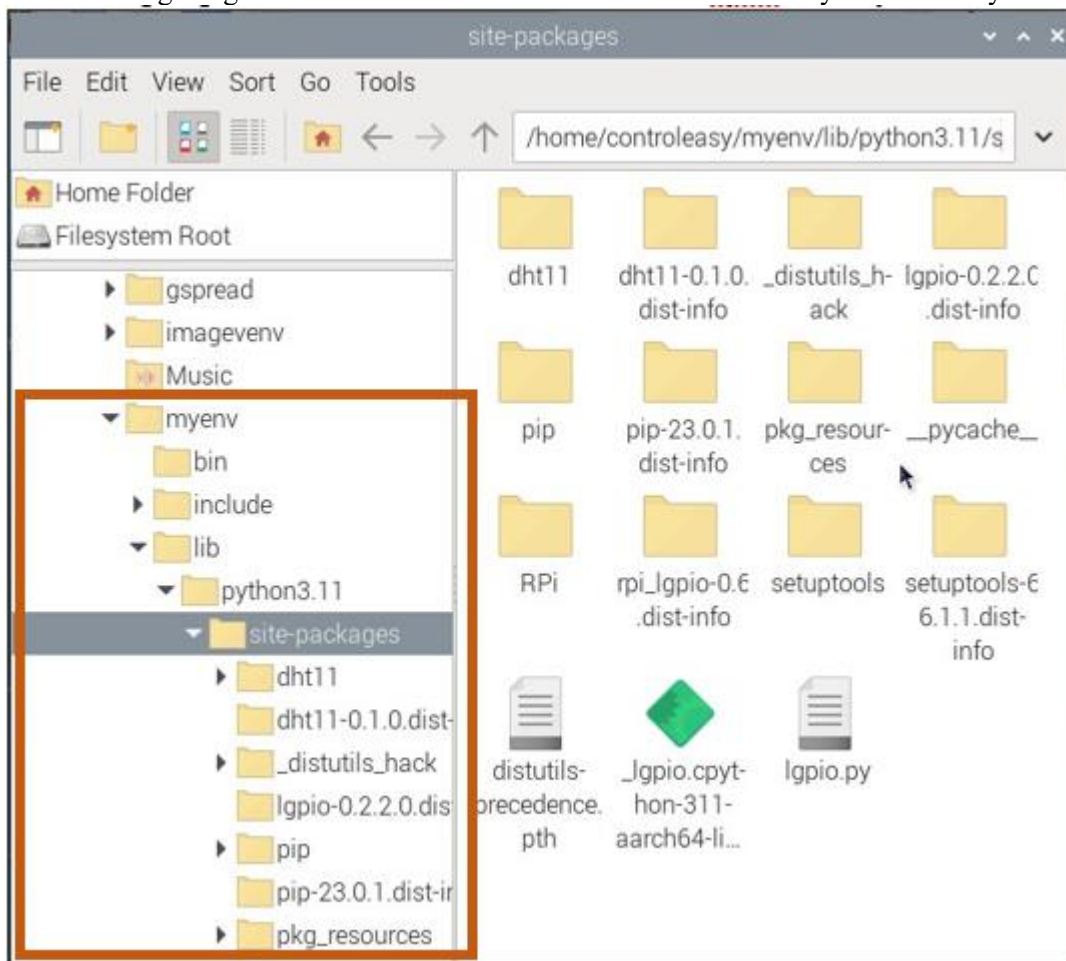
- c) After removing rpi.gpio library

```
(myenv) controleasy@raspberrypi:~ $ pip list
Package      Version
-----
dht11        0.1.0
pip          23.0.1
setuptools   66.1.1
```

- d) After installing rpi.lgpio library

```
(myenv) controleasy@raspberrypi:~ $ pip list
Package      Version
-----
dht11        0.1.0
lgpio        0.2.2.0
pip          23.0.1
rpi-lgpio    0.6
setuptools   66.1.1
```


The following image shows the installed modules' files inside the "myenv" directory:



You can find the location of your installed dht11 library by typing the following command in the terminal. (This is applicable for virtual environments and also system-wide environment):

```
controleasy@raspberrypi:~ $ python3 -m pip show dht11
```

Result:

```
controleasy@raspberrypi:~ $ python3 -m pip show dht11
Name: dht11
Version: 0.1.0
Summary: Pure Python library for reading DHT11 sensor on Raspberry Pi
Home-page: https://github.com/szazo/DHT11_Python
Author: Zoltán Szarvas
Author-email:
License:
Location: /usr/local/lib/python3.11/dist-packages/dht11-0.1.0-py3.11.egg
Requires: RPi.GPIO
Required-by:
controleasy@raspberrypi:~ $
```

What is ‘pip’ command?

‘pip’ is a command-line tool used to install and manage Python packages (libraries and modules). By default, ‘pip’ downloads packages from the Python Package Index (PyPI), which is a repository of thousands of Python packages.

During this process, ‘pip’ will:

- Install the package files into the environment's site-packages directory. When used in a virtual environment, this directory is specific to that environment.
- Resolve and install any dependencies specified in the package’s metadata (e.g., setup.py, pyproject.toml).
- Perform installation tasks as required by the package.

As of Debian 12 Bookworm OS, the ‘pip’ command cannot be used to install Python 3rd-party libraries into the root system anymore to ensure system integrity and stability. Instead, it is recommended to use virtual environments or user-specific installations for third-party packages.

2 PYTHON CODE FOR DHT11 MODULE

OK. Now let's take a look at the code we're going to use. We specifically used dht11 sensor for this tutorial.

THE COMPLETE PYTHON CODE FOR DHT11

Here is the complete python code for the DHT11 sensors. The sleep time in this code may be adjusted to be suitable for specific Raspberry Pi hardware in order to maximise sensor readings.

```
import RPi.GPIO as GPIO
import dht11
import time
import datetime

# Initialize GPIO settings
GPIO.setwarnings(True) # Enable GPIO warnings
GPIO.setmode(GPIO.BCM) # Use BCM (Broadcom) numbering for GPIO pins

# Initialize DHT11 sensor instance on GPIO pin 4
instance = dht11.DHT11(pin=4)

try:
    while True:
        # Read data from the DHT11 sensor
        result = instance.read()

        # Check if the reading is valid
        if result.is_valid():
            # Print the current timestamp, temperature, and humidity
            print("Last valid input: " + str(datetime.datetime.now()))
            print("Temperature: %-3.1f C" % result.temperature)
            print("Humidity: %-3.1f %% " % result.humidity)

            # Wait for 2 seconds before the next reading// next while True cycle
```

```
#(DHT11 requires at least 1 second between reads)

# Comment out this line if use Pi 3
time.sleep(2)

else:

    # Print the error code if the reading is invalid
    # Comment out this line for troubleshooting purposes
    # print("Error: %d" % result.error_code)

    #add a small delay before retrying to read sensor
    time.sleep(0.001)

except KeyboardInterrupt:

    # Cleanup GPIO settings before exiting the program
    print("Cleanup")
    GPIO.cleanup()
```

THE CODE EXPLANATIONS

The first section of code imports all necessary libraries or modules that the code needs.

```
import RPi.GPIO as GPIO
import dht11
import time
import datetime
```

`RPi.GPIO` is a gpio module that is used to interact with General Purpose Input/Output (GPIO) pins of the Raspberry Pi. This library lets you configure, read, and write to GPIO pins.

`Dht11` module depends on `RPi.GPIO`. The `RPi.GPIO` module is used by `dht11` library to setup a pin as Output Pin or Input Pin, and to write HIGH or LOW signal to the pin. By using `RPi.GPIO` module, `dht11` module able to read the raw signals from the DHT11 sensor. Then, `dht11` module converts the raw signals into meaningful data to obtain temperature and humidity values, and also checks for errors to ensure accurate readings

Time module is a low-level library, dealing with time in seconds and providing functions for sleeping, used for pausing the program for certain amount of time. Datetime module is a higher-level library, focusing on manipulating dates and times in a more human-readable format.

```
GPIO.setwarnings(True)
GPIO.setmode(GPIO.BCM)
```

Now, we use the gpio module to enables warnings if there is any issue in our GPIO pin. The warnings may occur when we are reusing pins: trying to use a previously used pin without cleaning up “`GPIO.cleanup()`”, or when we use incorrect pin configurations (e.g., trying to set a pin as an output when it was previously set as an input). Setting it to True means that warnings will be shown, which is useful for debugging.

Then, we set the gpio pin numbering mode. `GPIO.setmode()` determines how you refer to the GPIO pins in your code. There are two primary modes: BCM Mode and Board Mode. In BCM mode, “`GPIO.setmode(GPIO.BCM)`”, you refer to the GPIO pins by their Broadcom chip-specific pin numbers. Whereas in BOARD Mode, “`GPIO.setmode(GPIO.BOARD)`”, you refer to the GPIO pins by their physical pin numbers on the Raspberry Pi's header.

```
instance = dht11.DHT11(pin=4)
```

The following line creates an instance of the DHT11 class, initializing the DHT11 sensor. The pin=4 argument specifies that the sensor is connected to GPIO pin 4.

```
try:
    # Code that might raise an exception
    ...
    ...
except KeyboardInterrupt:
    # Cleanup GPIO settings before exiting the program
    print("Cleanup")
    GPIO.cleanup()
```

A “try” block in Python is used to handle exceptions that may occur during the execution of a block of code. An **exception** is an event that disrupts the normal flow of execution in a program, such as errors or unusual conditions that arise during runtime, which the program needs to handle to avoid crashing or behaving unexpectedly. A “try” allows you to catch and manage errors gracefully without terminating the program abruptly.

In our case, we use “except KeyboardInterrupt” in a case where users are to stop the program manually by pressing “Ctrl+C”. Without the try block, pressing Ctrl+C would abruptly terminate the script, potentially leaving GPIO pins in an undefined state. By using “except KeyboardInterrupt”, the program safely resets the GPIO pins to their default state before exiting the program.

We use RPi.GPIO module’s GPIO.cleanup() to reset all GPIO pins that our program used to their default state. This is important to prevent conflicts with other programs or scripts that might use the GPIO pins later.

```
while True:
    result = instance.read()
    if result.is_valid():
        print("Last valid input: " + str(datetime.datetime.now()))
        print("Temperature: %-3.1f C" % result.temperature)
        print("Humidity: %-3.1f %% " % result.humidity)
        time.sleep(2)
    else:
        print("Error: %d" % result.error_code)
        time.sleep(0.001)
```

Now, we will look at the main part of the code. “while True:” creates an infinite loop that will keep running until it is interrupted (that is., by pressing Ctrl+C or another stop condition).

“result = instance.read()” reads the temperature and humidity data from the DHT11 sensor and then assigns them to “result” variable. The “is_valid()” method returns True if the data is valid, and False otherwise.

If the data read is valid, the program prints the current date and time, the temperature value, and the humidity value. After that, the program is paused for a few seconds, using “time.sleep()”, before the next iteration of the “while True” loop. This is because DHT11 sensor requires at least a 1-second delay between readings. If the data is not valid, the program prints the error code returned by the dht11 module, and then retry the “while True” loop after a very short delay.

3 RUN THE PROGRAM

Now, you can run the program depending on where you install all the libraries.

i) **Method 1: Virtual environment**

- 1) Open your virtual environment.

```
controleasy@raspberrypi:~ $ source myenv/bin/activate
```

- 2) Navigate to the directory where you save the Python program.

```
(myenv) controleasy@raspberrypi:~ $ cd /home/controleasy/Desktop/Pi5compatible
```

- 3) Run the program

```
(myenv) controleasy@raspberrypi:~/Desktop/Pi5compatible $ python rpidht-tested.py
Last valid input: 2024-08-14 04:06:02.010602
Temperature: 22.3 C
Humidity: 68.0 %
Last valid input: 2024-08-14 04:06:03.585923
Temperature: 22.2 C
Humidity: 67.0 %
Last valid input: 2024-08-14 04:06:05.160997
Temperature: 22.2 C
Humidity: 67.0 %
Last valid input: 2024-08-14 04:06:06.736101
Temperature: 22.2 C
Humidity: 67.0 %
```


ii) Method 2: System-wide environment

If your libraries are installed in system-wide environment, you can run the code from the terminal or directly through any code editor software.

Using the terminal


- 1) Navigate to the directory where you save the Python program.

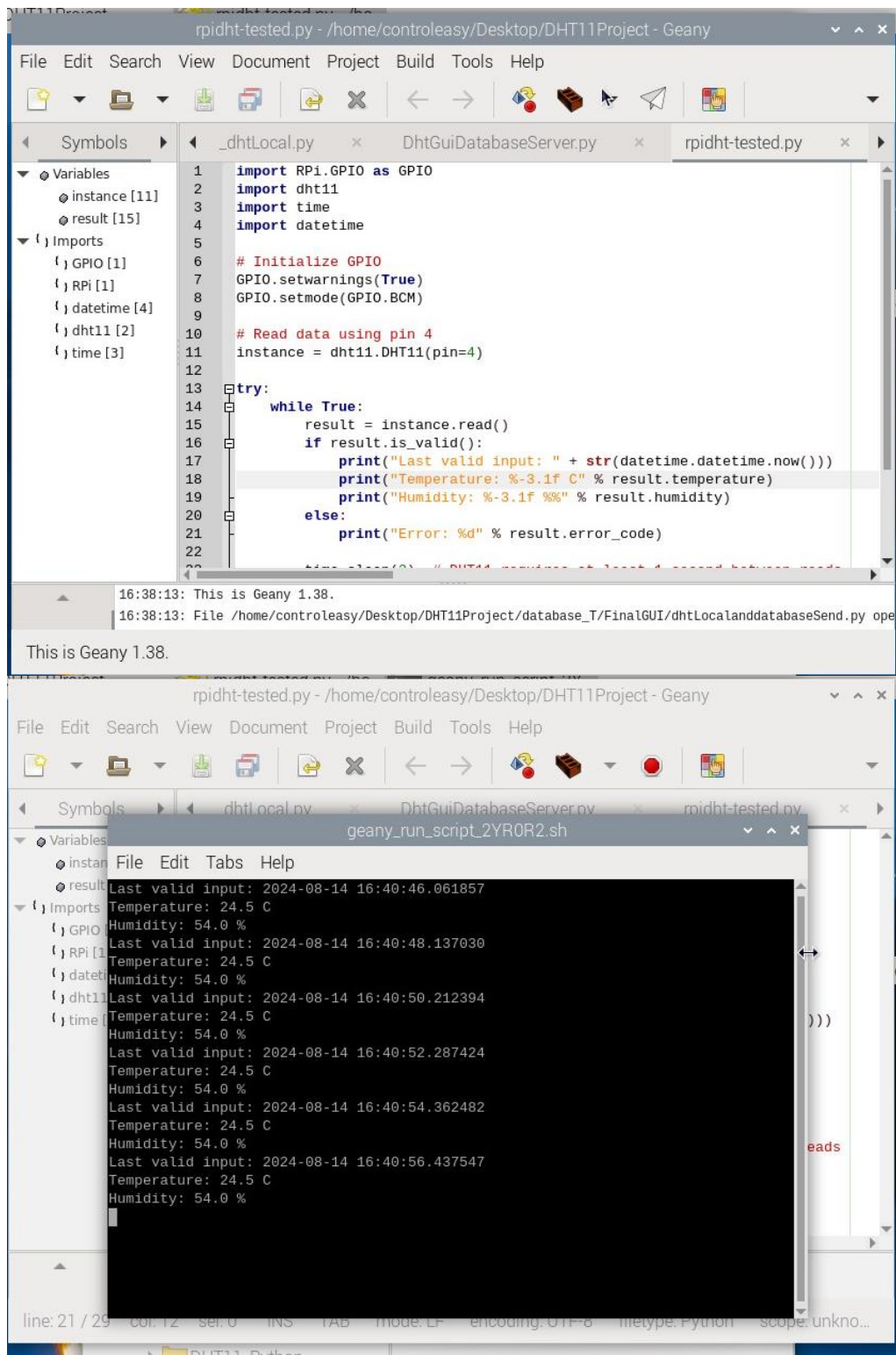
```
controleasy@raspberrypi:~ $ cd /home/controleasy/Desktop/DHT11Project
```

- 2) Run the program

```
controleasy@raspberrypi:~/Desktop/DHT11Project $ python rpidht-tested.py
Last valid input: 2024-08-14 16:42:33.627631
Temperature: 24.8 C
Humidity: 54.0 %
Last valid input: 2024-08-14 16:42:35.702715
Temperature: 24.5 C
Humidity: 54.0 %
Last valid input: 2024-08-14 16:42:37.777755
Temperature: 24.5 C
Humidity: 54.0 %
Last valid input: 2024-08-14 16:42:39.852799
Temperature: 24.5 C
Humidity: 54.0 %
Last valid input: 2024-08-14 16:42:41.927756
Temperature: 24.5 C
Humidity: 54.0 %
Last valid input: 2024-08-14 16:42:44.002884
Temperature: 24.5 C
Humidity: 54.0 %
```

Using Geany Software

- 1) Load Geany Programmer's Editor
- 2) Click File > Open and then navigate to your Python program
- 3) You can run the program by clicking Build > Execute, by pressing F5 on the keyboard, or by pressing the following button: 

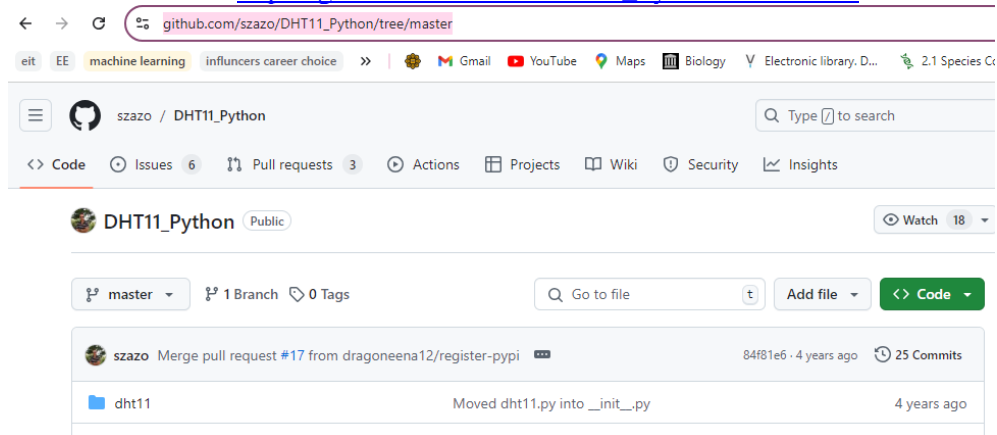


4 IMPROVING DHT11 LIBRARY

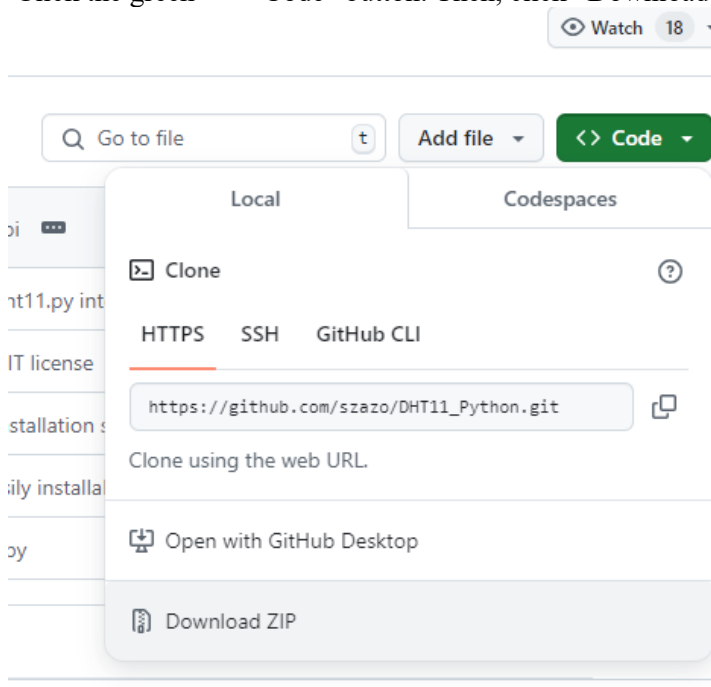
You may notice the output result for Raspberry Pi 4 and Pi 3 are not satisfactory. This is because the dht11 library have some weaknesses. In this section, we will make a little edit on the dht11 library so that the communication between dht11 sensor and Raspberry Pi can be improved.

4.1 Get dht11 into your project directory

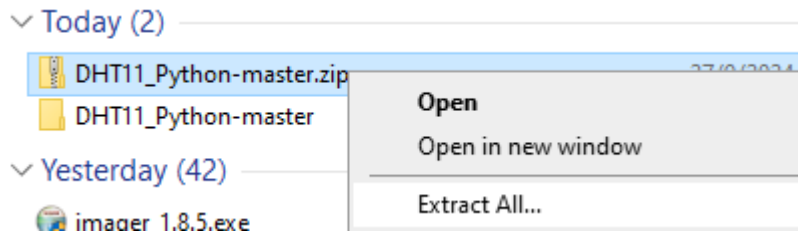
- 1) Go to this website: https://github.com/szazo/DHT11_Python/tree/master



- 2) Click the green “<> Code” button. Then, click “Download ZIP”



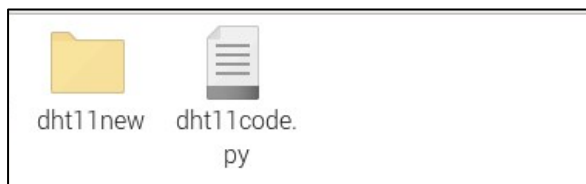
- 3) Extract the downloaded ZIP file



- 4) Go inside the extracted folder. Inside it, copy only “dht11” folder. Locate “dht11” folder to your Raspberry Pi, inside the same directory as your program code.

File name	Date modified	Type	Size
dht11	27/9/2024 10:18 AM	File folder	
example.py	27/9/2024 10:18 AM	Python File	1 KB
LICENSE.md	27/9/2024 10:18 AM	Markdown Source...	2 KB
README.md	27/9/2024 10:18 AM	Markdown Source...	1 KB
setup.py	27/9/2024 10:18 AM	Python File	1 KB

- 5) Rename the file as “dht11new”



4.2 Edit dht11 program.

- 1) In your program, you must only edit one line.

Change “import dht11” to “import dht11new as dht11”

Before:

```
1 import RPi.GPIO as GPIO
2 import dht11
3 import time
4 import datetime
5
6 # Initialize GPIO settings
```

After:

```
1 import RPi.GPIO as GPIO
2 import dht11new as dht11
3 import time
4 import datetime
5
```

- 2) Save the change.

4.3 Edit dht11 library

- 1) Open the “__init__.py” file located inside the “dht11new” folder.
- 2) Find and modify these 2 numbers: 0.05 become 0.5, and 0.02 become 0.018

Before:

```
32
33 def read(self):
34     RPi.GPIO.setup(self.__pin, RPi.GPIO.OUT)
35
36     # send initial high
37     self.__send_and_sleep(RPi.GPIO.HIGH, 0.05)
38
39     # pull down to low
40     self.__send_and_sleep(RPi.GPIO.LOW, 0.02)
41
42     # change to input using pull up
43     RPi.GPIO.setup(self.__pin, RPi.GPIO.IN, RPi.GPIO.PUD_UP)
44
```

After:

```
def read(self):
    RPi.GPIO.setup(self.__pin, RPi.GPIO.OUT)

    # send initial high
    self.__send_and_sleep(RPi.GPIO.HIGH, 0.5)

    # pull down to low
    self.__send_and_sleep(RPi.GPIO.LOW, 0.018)

    # change to input using pull up
    RPi.GPIO.setup(self.__pin, RPi.GPIO.IN, RPi.GPIO.PUD_UP)
```

- 3) Find “__send_and_sleep” function. Then modify them as follows.

Before:

```
def __send_and_sleep(self, output, sleep):
    RPi.GPIO.output(self.__pin, output)
    time.sleep(sleep)
```

After:

```
78
79 def __send_and_sleep(self, output, sleep):
80     RPi.GPIO.output(self.__pin, output)
81
82     start_time = time.perf_counter()
83     while time.perf_counter() - start_time < sleep:
84         pass
85
```

- 4) Save all the changes in the library.

- 5) Now, you can run your sensor code in your preferred environment. The output of dht11 increases drastically with this improvement.

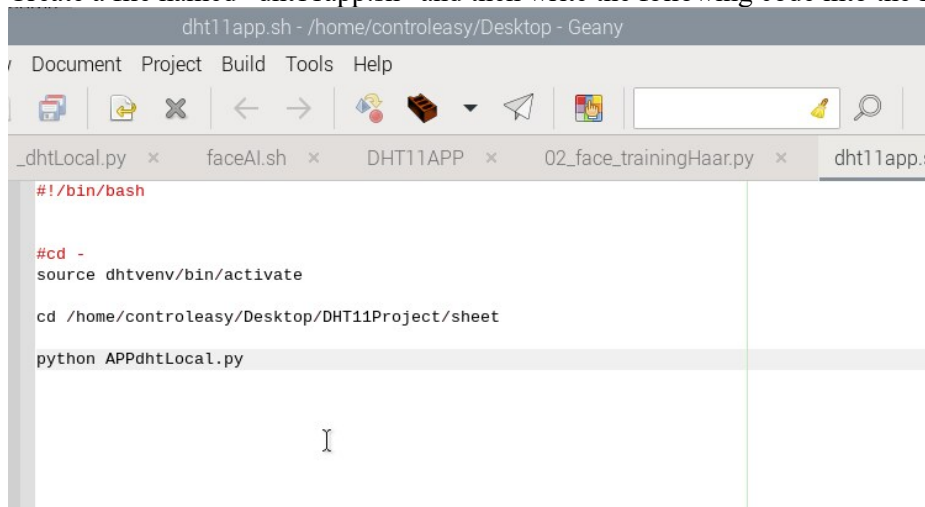
```
(dhtenv) controleasy@raspberrypi:~/Desktop $ python dht11code.py
Last valid input: 2024-09-27 10:05:20.184125
Temperature: 26.7 C
Humidity: 59.0 %
Last valid input: 2024-09-27 10:05:22.707501
Temperature: 26.2 C
Humidity: 58.0 %
Last valid input: 2024-09-27 10:05:25.230960
Temperature: 26.2 C
Humidity: 58.0 %
Last valid input: 2024-09-27 10:05:27.754276
Temperature: 26.2 C
Humidity: 58.0 %
Last valid input: 2024-09-27 10:05:30.277656
Temperature: 26.2 C
Humidity: 58.0 %
Last valid input: 2024-09-27 10:05:32.800906
Temperature: 26.2 C
Humidity: 58.0 %
Last valid input: 2024-09-27 10:05:35.324146
Temperature: 26.2 C
Humidity: 58.0 %
Last valid input: 2024-09-27 10:05:37.847487
```

5 ADDITIONAL INFO – MAKING EXECUTABLE APP IN RASPBERRY PI

In the previous sections, whenever we wish to run the code in virtual environment, we need to access the terminal, navigate to appropriate directories, then, run the code. This process may become tedious after a while.

An alternative solution is to create an executable app which will run the python code inside a specific virtual environment automatically for us.

- 1) Create a file named “dht11app.sh” and then write the following code into the file:

A screenshot of a code editor window titled "dht11app.sh - /home/controleasy/Desktop - Geany". The editor has a menu bar with "Document", "Project", "Build", "Tools", and "Help". Below the menu is a toolbar with various icons. The editor shows several tabs: "_dhtLocal.py", "faceAI.sh", "DHT11APP", "02_face_trainingHaar.py", and "dht11app.sh". The active tab "dht11app.sh" contains the following code:

```
#!/bin/bash

#cd -
source dhtvenv/bin/activate

cd /home/controleasy/Desktop/DHT11Project/sheet

python APPdhtLocal.py
```

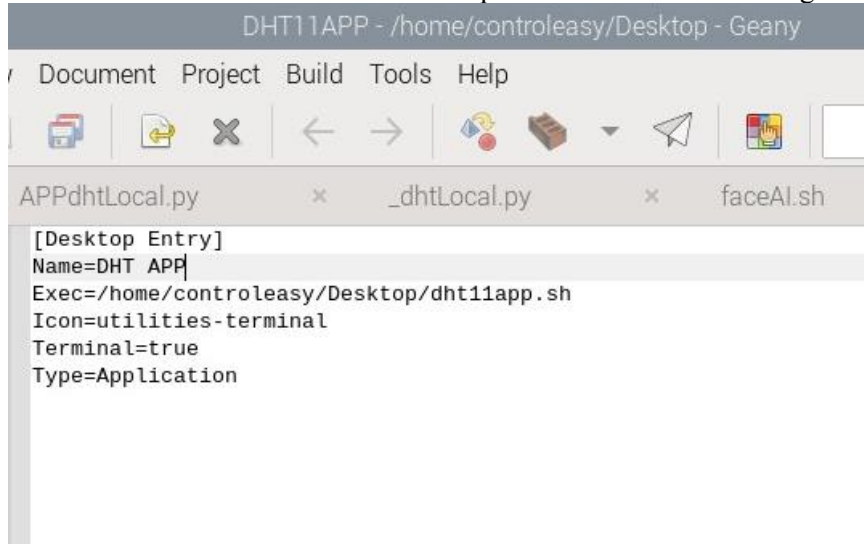
The first line activates a specific virtual environment named “dhtvenv”. The second line navigates to the directory where the python code is kept, “/home/controleasy/Desktop/DHT11project/sheet”. The third line is for running the python code file, that is, “APPdhtLocal.py”.

You can change these scripts depending on your project.

- 2) Make the file executable by writing this command in the terminal

```
$ chmod +x dht11app.sh
```

- 3) Create a file named “DHT11APP.desktop”. Then write the following lines inside the file:



“Name” is the name that will appear on the desktop, you can put any name you preferred.

“Exec” refers to the location of the “dht11app.sh” file created initially. “Terminal = true” means the application will run in a terminal; “Terminal = false” means that it will not run in a terminal. For “Icon” section, you may write the path to your specific icon file instead of using “utilities-terminal”.

- 4) Save the “.desktop” file on your desktop
- 5) Make sure the “.desktop” file is executable by writing the following command to the terminal:

```
$ chmod +x DHT11APP.desktop
```


- 6) Now, you can just double-click the “.desktop” file to run the python script in its corresponding virtual environment.

