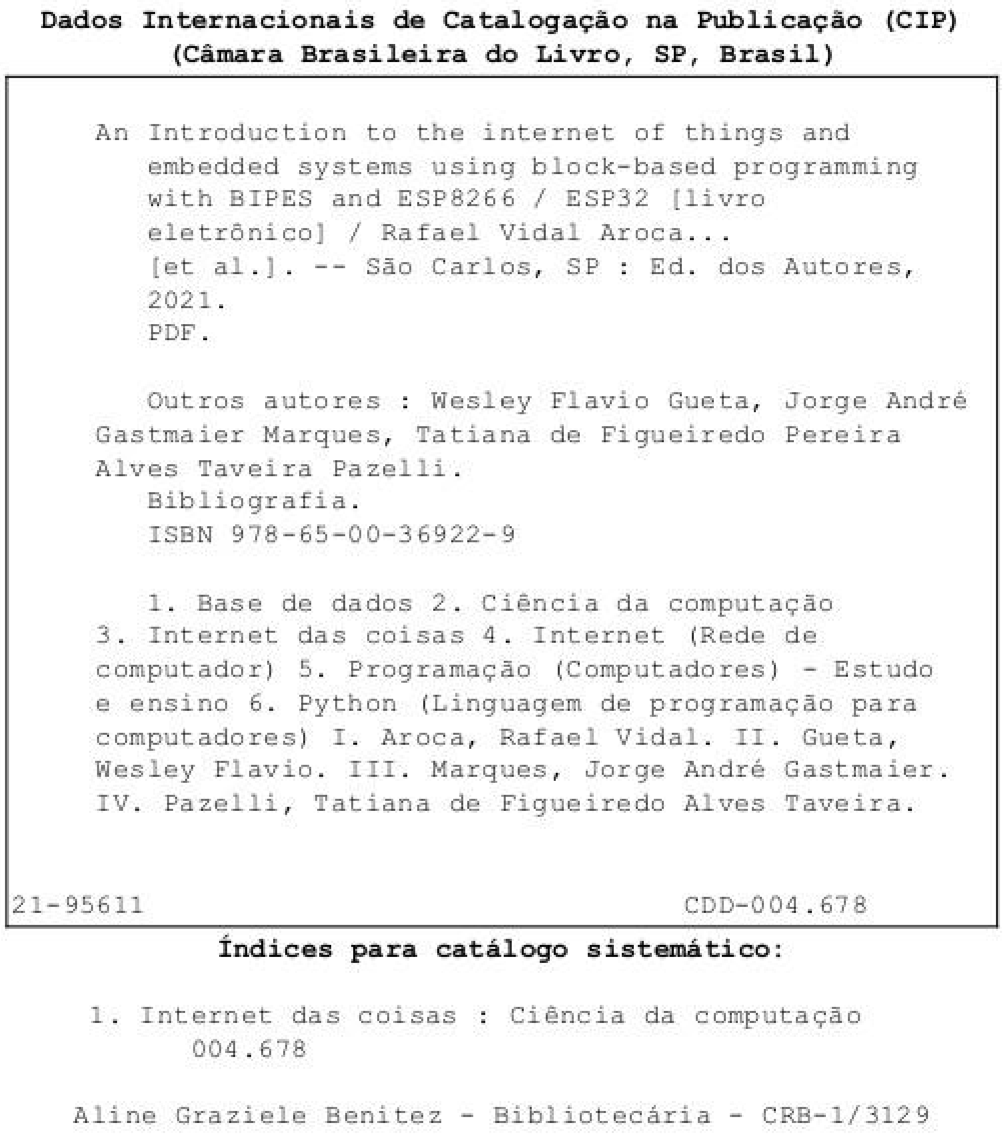
BIPES

An introduction to the Internet of Things and Embedded Systems using block-based programming with BIPES and ESP8266 / ESP32

Rafael Vidal Aroca
Wesley Flavio Gueta
Jorge André Gastmaier Marques
Tatiana Figueiredo Pereira Alves Taveira Pazelli

December 2021 - 1st Edition

Index Card



Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

An Introduction to the internet of things and embedded systems using block-based programming with BIPES and ESP8266 / ESP32 [livro eletrônico] / Rafael Vidal Aroca... [et al.]. -- São Carlos, SP : Ed. dos Autores, 2021.
PDF.

Outros autores : Wesley Flavio Gueta, Jorge André Gastmaier Marques, Tatiana de Figueiredo Pereira Alves Taveira Pazelli.
Bibliografia.
ISBN 978-65-00-36922-9

1. Base de dados 2. Ciência da computação 3. Internet das coisas 4. Internet (Rede de computador) 5. Programação (Computadores) - Estudo e ensino 6. Python (Linguagem de programação para computadores) I. Aroca, Rafael Vidal. II. Gueta, Wesley Flavio. III. Marques, Jorge André Gastmaier. IV. Pazelli, Tatiana de Figueiredo Alves Taveira.

21-95611 CDD-004.678

Índices para catálogo sistemático:

1. Internet das coisas : Ciência da computação 004.678

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

Table of Contents

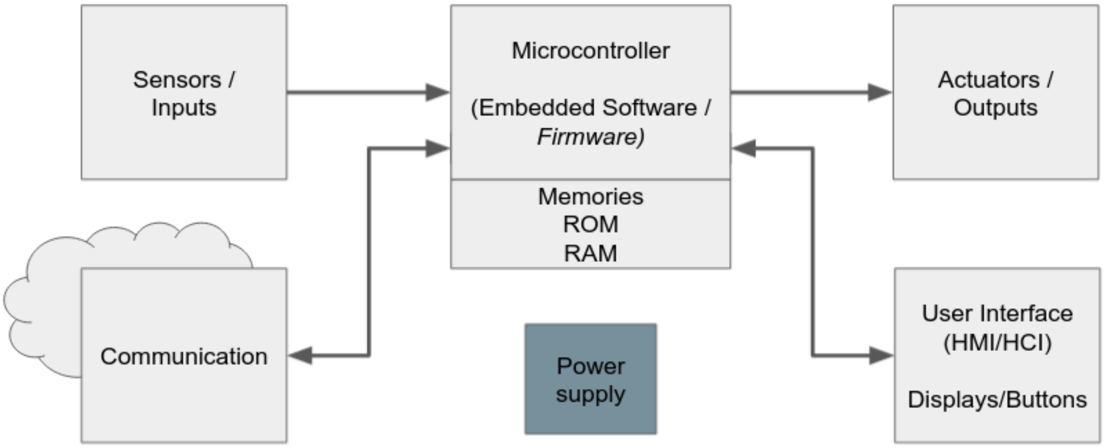
Internet of things using BIPES

Quick Introduction

BIPES: Block-based Integrated Platform for Embedded Systems (http://bipes.net.br) is an open-source and free platform created for embedded systems programming and Internet of Things applications [1].

Embedded systems are present in every part of everyday life: in commerce, industry, cars, ovens, televisions, copiers, garage doors, alarms, refrigerators, and other devices. An embedded system is composed of hardware and software that perform a specific task and is “embedded” in a product. Hence the names: embedded system, embedded hardware, and embedded software. Several devices are available to implement embedded systems, such as microprocessors and microcontrollers. Microcontrollers, in particular, are small integrated circuits (chips), usually of low cost, that need few external components and can implement an operating logic based on inputs and other information to control outputs and perform other actions. They are a kind of small and complete computer designed for specific applications. Embedded software, in general, is what determines the operating logic of an embedded system.

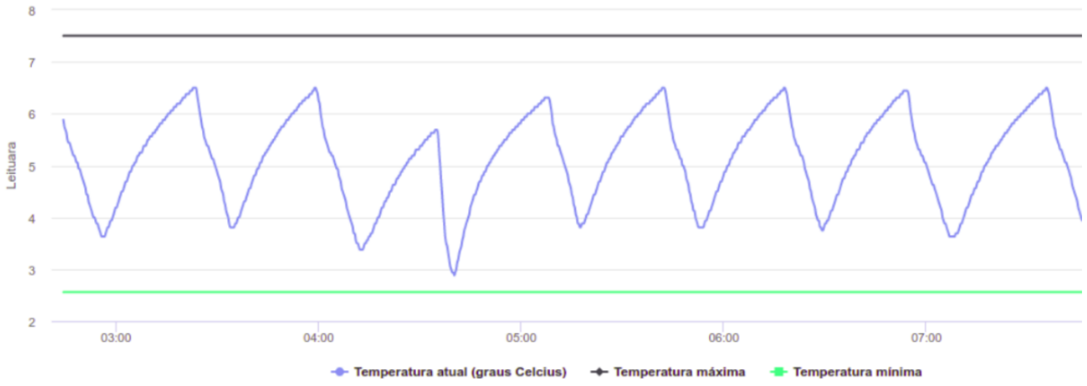
The figure below shows an overview of an embedded system composed of a microcontroller and several other devices. However, not all of these components are necessary for all applications. Thus, each application will determine the need for sensors, actuators, and other elements in an embedded system.



- **Microcontroller**: It is an electronic component, usually composed of input and output ports for electrical signals, by memories and a processor capable of executing the logic defined by the embedded software or *firmware*;

- **Sensors**: Sensors are devices that detect/measure the state/level of a specific physical magnitude of the environment, such as temperature, humidity, lighting, vibration, magnetic field, among others;
- **Actuators**: Actuators are devices that can change the state of the environment, such as motors, heaters, electromagnets, and other devices that can, through microcontroller commands, act on the environment. The majority of microcontrollers cannot directly drive an actuator, so power circuits (*drivers)*, which can include transistors or H-bridges, must be used so that microcontrollers can drive actuators;
- **Communication**: Some embedded systems may include communication mechanisms, allowing them to send data to remote monitoring systems or receive remote commands. This type of connectivity can be wired or wireless, and it may or may not have an Internet connection;
- **Power supply**: An embedded system requires a power source to provide power for its operation. The energy can be obtained from the mains, batteries, solar panels, among other sources of electrical energy. A trend in some embedded systems and Internet of Things systems are systems that harvest energy from the environment, for example, from radio waves;
- **User interface**: Several embedded systems need to interact with the user, whether through displays, buttons, keyboards, voice messages, voice commands, or even through control and visualization interfaces based on smartphones.

Considering these definitions, the figure below shows an example of the result of the operation of an embedded system composed of a microcontroller, a temperature sensor, and a compressor (actuator) controlling the internal temperature of a refrigerator. The temperature sensor and the microcontroller cost less than a hamburger, and it is possible to implement different logic and functionalities. In this figure, you can see that every time the temperature goes above 6 degrees Celsius, something happens, and the temperature starts to drop. The onboard system detects that the temperature has exceeded 6 degrees Celsius and activates the refrigerator's compressor, which cools the environment. Consequently, the temperature sensor also detects this change in the environment. Upon reaching the temperature of 4 degrees, the actuator is turned off. Thus, in this application, the cycle is repeated in a control action known as "*on-off*", following the logic implemented in the embedded software.



This system could also be connected to the Internet, allowing the desired temperature setting to be monitored and configured remotely using a smartphone. In addition, the system could issue alerts and alarms in the event of compressor failure or when an inappropriate temperature threshold is reached. This association of an embedded system, with user inputs and monitoring via the Internet, is one example of an ***Internet of Things*** (IoT) system. Thus, thousands of devices are connected to the Internet, such as light bulbs, smart meters, refrigerators, air conditioners, home assistants, and others.

Still, it is impressive to analyze the advances in recent years in the context of embedded systems. This book presents hands-on activities using the ESP8266 or the ESP32 module, both from Espressif Systems. These modules, classified as *Systems on Chips (SoCs),* are complete computers, including processor, RAM, FLASH memory to store programs, WiFi wireless connection, sensors' inputs, actuators' outputs, and various other features. And they cost less than a hamburger! If you think about it, they are much more powerful than the computer that brought the Apollo Lunar Module to the Moon's surface. See the figure below for a comparison.

Although the figure doesn't show the ESP32 module in the comparison, it is a device more powerful than the ESP8266 with several additional features, including a dual-core processor, Bluetooth, increased memory and processing capacity, and other features. It is important to note that all explanations and activities presented in this book are illustrated with examples for the ESP8266, which work equally on the ESP32, occasionally with a few differences, such as changing the pins numbers because the two modules have different pin input and output configurations (electronic terminals).

Apollo Guidance Computer

Clock: 2MHz
RAM: 2KB
ROM: 36KB
Weight: 32000 grams
Power: 55 Watts
Price: US$ 150000,00

Arduino UNO

Clock: 16MHz
RAM: 2KB
ROM: 32KB
Weight: 25 grams
Power: 0,2 Watts
Price: US$ 7,00

ESP8266

Clock: 160MHz
RAM: ~ 100KB
ROM / Flash: 16MB
Weight: 1.5 grams
Power: 0,3 Watts
Price: ~US$ 1,00

Wifi / MicroPython + BIPES
www.bipes.net.br

BIPES is used for all activities proposed in this book. Versatile, open, and free, the BIPES platform allows programming using several microcontroller boards such as ESP32, ESP8266, Arduino, mBed, among others. Furthermore, the **BIPES** can be used directly in web browsers for multiple devices and does not require installing any software, plugin, or setting a particular configuration. In addition to the ease of development through block programming, **BIPES** also offers an Internet of Things platform, enabling the construction of visualization and remote control panels (dashboards) for IoT applications [2].

Although block-based programming may appear to be focused on educational applications due to its ease, it has been adopted and used each time more in commercial and industrial environments. Several advantages have already been noticed: reduction of development time, decrease in logic failures, greater ease of understanding, and quicker and easier maintenance. A publication by the Institute of Electrical and Electronics Engineers (IEEE) released an article talking about the birth of the era of programming without code [3].

As mentioned, this is a quick introduction to the topic. Several quality publications are available for you to delve into this subject. The focus here is to guide you through quick and easy activities to implement IoT systems practically.

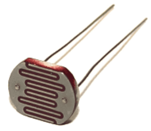
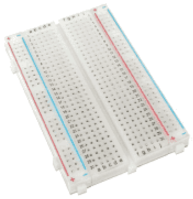
References:

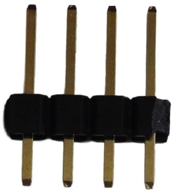
[1] AGDS Junior, LMG Gonçalves, GA de Paula Caurin, GTB Tamanaka, AC Hernandes, RV Aroca. BEEPS: Block Based Integrated Platform for Embedded Systems. **IEEE Access**, v. 8, p. 197955-197968, 2020.
Available from: <https://ieeexplore.ieee.org/document/9246562>.

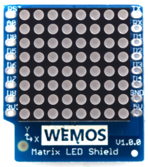
[2] CA Silva. **Desenvolvimento e validação de módulo de comunicação MQTT para plataforma BIPES para aplicações de Internet das Coisas.** 2020. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) - Universidade Federal de São Carlos, 2020.
Available at: <https://repositorio.ufscar.br/handle/ufscar/13656>.

[3] RD Caballar. Programming without code: The rise of no-code software development. **IEEE Spectrum**. Tech Talks, Nov. 2020, Disponível:
<https://spectrum.ieee.org/programming-without-code-no-code-software-development>

Bill of Materials

Item		**Description**
1		Any board with ESP8266 or ESP32 module. Specifically, all examples in this book used the Wemos D1 Mini module, based on the ESP8266 SoC
2		DHT11 Humidity and Temperature Sensor
3		Light Dependent Resistor sensor (LDR)
4		Prototyping board - protoboard
5		AC electric plug
6		Board with relay
7		AC electric plug receptacle

8		AC plug adapter for light bulb/lamp
9		Terminals
10		*Jumper cables*
11		*Power bank*
12		127 Volts AC Light build
13		Potentiometer
14		Parallel cable (white) for connection to the electrical network

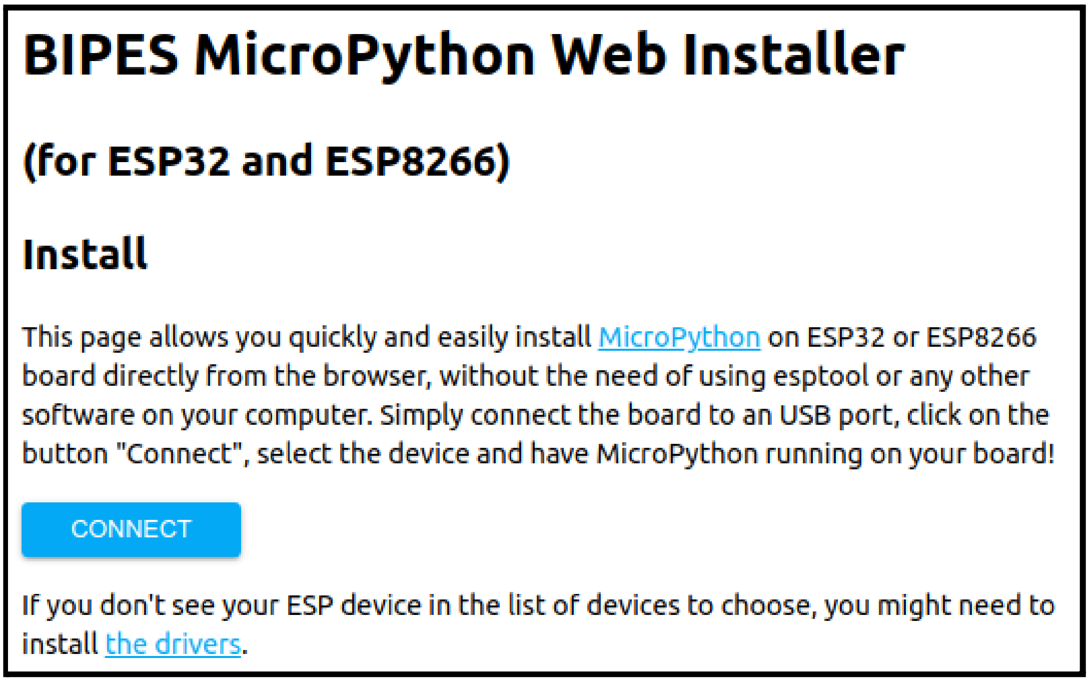
15		Switch (button) / Push Button
16		Micro servo motor / RC Hobby Servo
17		Different resistors with values of 1K, 570 ohms, and 220 ohms
18		*Buzzer* (piezoelectric sound emitter)
19		Light-emitting diode (LED)
20		3-color RGB Light-emitting diode (LED) Optional: only for the activity “Changing color weather rooster”
21		LED Matrix Shield with TM1640 controller Optional: only for the second part of the activity “Changing color weather rooster”

Preparation of the ESP8266 or ESP32 board

Several boards can carry out the activities proposed in this text. However, we will focus on using the board with the ESP8266 module on this occasion. Furthermore, MicroPython (https://micropython.org/) must be installed on this board, allowing you to program, debug and manage applications practically and dynamically, with a minimalist implementation of the Python programming language. Using the resources of the BIPES project, installing MicroPython on the ESP8266 is quite simple. The following steps work on Google Chrome and Microsoft Edge browsers in MAC, Linux, and Windows environments.

To install MicroPython on ESP8266:

1. Go to: https://bipes.net.br/flash/esp-web-tools/.
2. Connect the ESP8266 board to the computer's USB port.
3. Click **CONNECT**.



BIPES MicroPython Web Installer

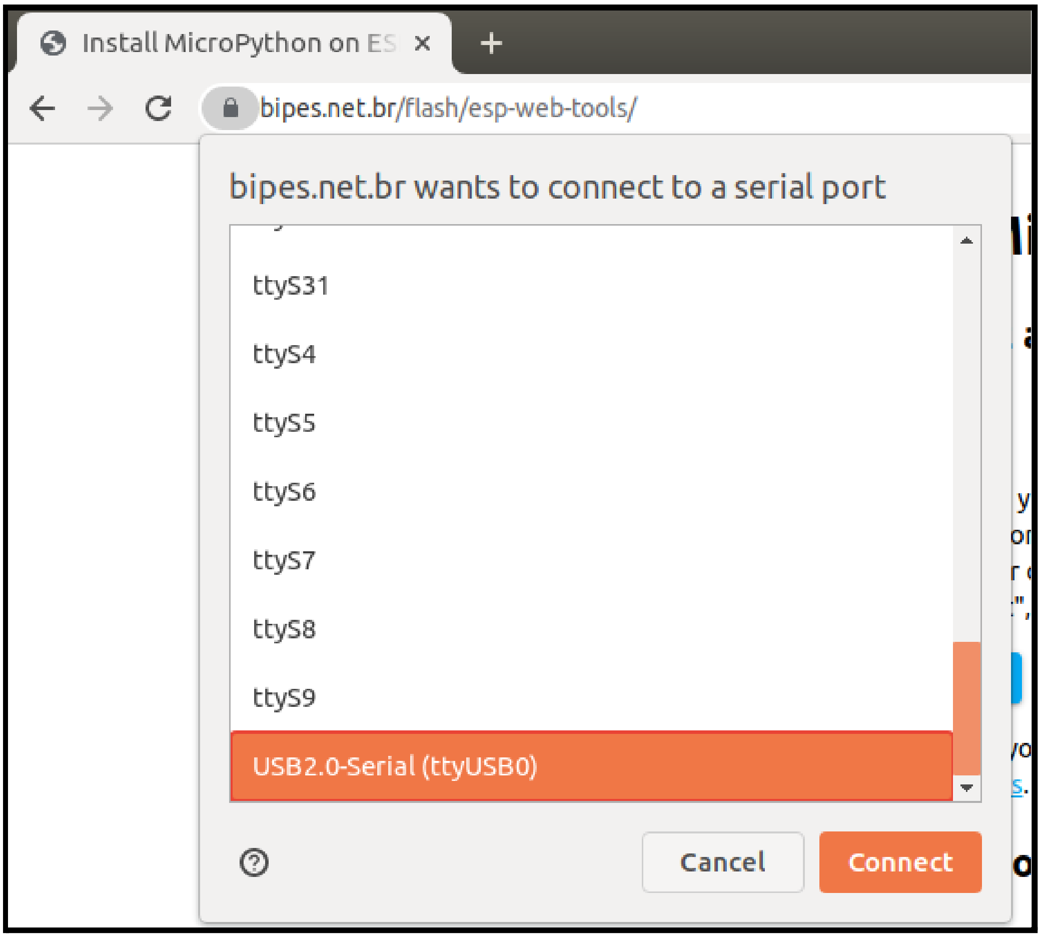
(for ESP32 and ESP8266)

Install

This page allows you quickly and easily install MicroPython on ESP32 or ESP8266 board directly from the browser, without the need of using esptool or any other software on your computer. Simply connect the board to an USB port, click on the button "Connect", select the device and have MicroPython running on your board!

If you don't see your ESP device in the list of devices to choose, you might need to install the drivers.

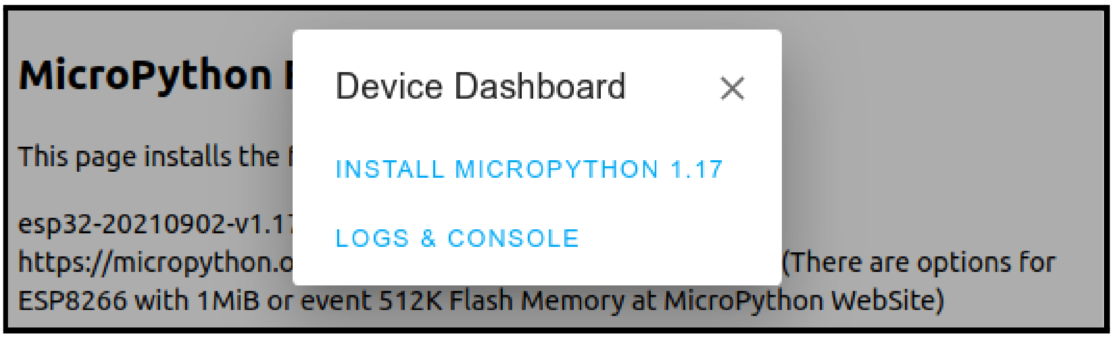
4. Select the serial (COM) port used by your device and click **Connect**:



bipes.net.br wants to connect to a serial port

ttyS31
ttyS4
ttyS5
ttyS6
ttyS7
ttyS8
ttyS9
USB2.0-Serial (ttyUSB0)

5. Then click **INSTALL** MICROPYTHON:

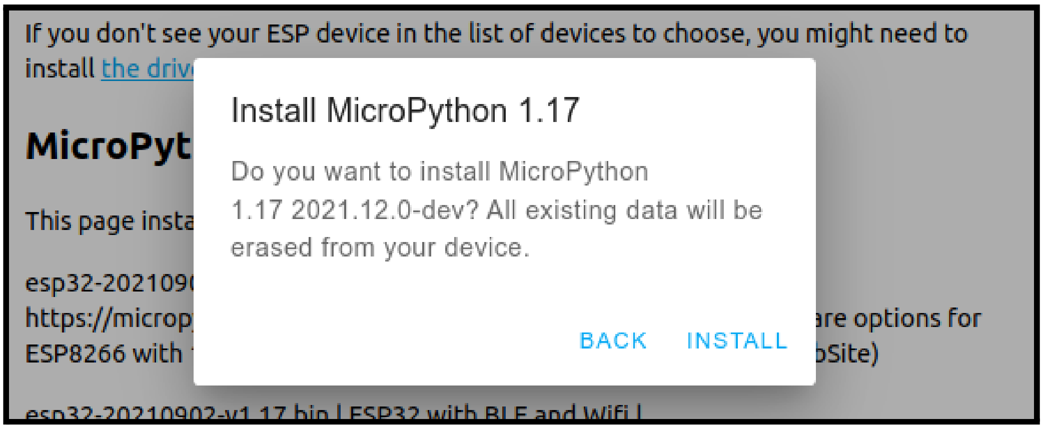


MicroPython I

This page installs the I

esp32-20210902-v1.17
https://micropython.o (There are options for ESP8266 with 1MiB or event 512K Flash Memory at MicroPython WebSite)

6. Then click **INSTALL**:



If you don't see your ESP device in the list of devices to choose, you might need to install the driv

MicroPyt

This page insta

esp32-202109
https://microp
ESP8266 with

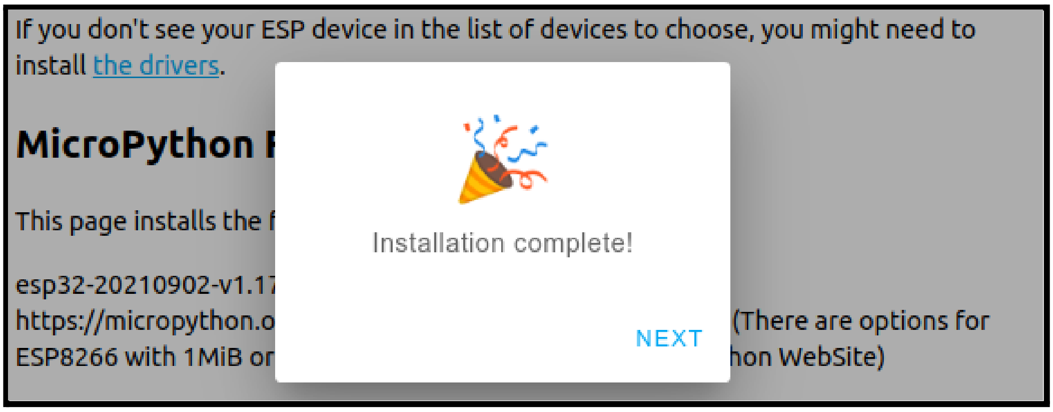
are options for
bSite)

esp32-20210902-v1.17.bin | ESP32 with BLE and Wifi |

Install MicroPython 1.17

Do you want to install MicroPython 1.17 2021.12.0-dev? All existing data will be erased from your device.

7. After installation, you will see the message:



If you don't see your ESP device in the list of devices to choose, you might need to install the drivers.

MicroPython F

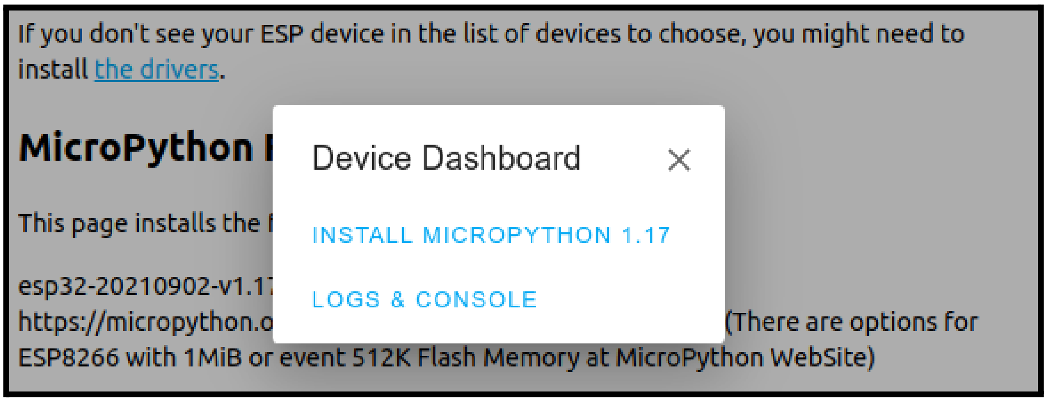
This page installs the f

esp32-20210902-v1.17
https://micropython.o
ESP8266 with 1MiB or

(There are options for
hon WebSite)

Installation complete!

8. Click **NEXT** and then **LOGS & CONSOLE**.



If you don't see your ESP device in the list of devices to choose, you might need to install the drivers.

MicroPython I

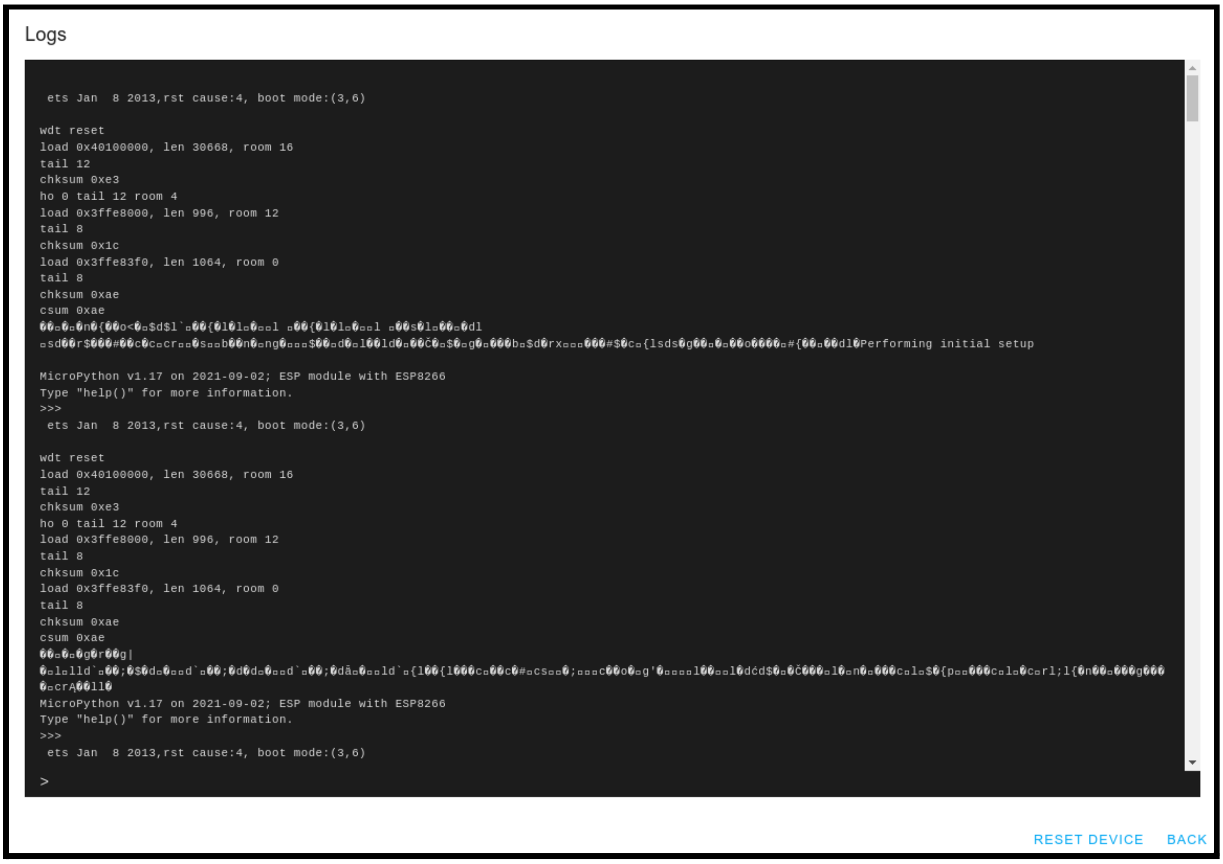
This page installs the f

esp32-20210902-v1.17
https://micropython.o
ESP8266 with 1MiB or event 512K Flash Memory at MicroPython WebSite)

(There are options for

Device Dashboard

Observe the behavior of the board on the console. By default, MicroPython creates a “MicroPython” network for remote connection and programming via WiFi. However, some generic ESP8266 boards have power circuits that provide insufficient electrical current to supply the board’s energy demand when the ESP8266 board starts the WiFi Access Point network service. As a result, the board may begin to restart without stopping (as shown in the image below).



```
 ets Jan  8 2013,rst cause:4, boot mode:(3,6)

wdt reset
load 0x40100000, len 30668, room 16
tail 12
chksum 0xe3
ho 0 tail 12 room 4
load 0x3ffe8000, len 996, room 12
tail 8
chksum 0x1c
load 0x3ffe83f0, len 1064, room 0
tail 8
chksum 0xae
csum 0xae
��▫�▫�n�{��o<�▫$d$l`▫��{�l�l▫�▫▫l ▫��{�l�l▫�▫▫l ▫��s�l▫��▫�dl
▫sd��r$���#��c�c▫cr▫▫�s▫▫b��n�▫ng�▫▫▫$��▫d�▫l��ld�▫��Č�▫$�▫g�▫���b▫$d�rx▫▫▫���#$�c▫{lsds�g��▫�▫��o����▫#{��▫��dl�Performing initial setup

MicroPython v1.17 on 2021-09-02; ESP module with ESP8266
Type "help()" for more information.
>>>
 ets Jan  8 2013,rst cause:4, boot mode:(3,6)

wdt reset
load 0x40100000, len 30668, room 16
tail 12
chksum 0xe3
ho 0 tail 12 room 4
load 0x3ffe8000, len 996, room 12
tail 8
chksum 0x1c
load 0x3ffe83f0, len 1064, room 0
tail 8
chksum 0xae
csum 0xae
��▫�▫�g�r��g|
�▫l▫lld`▫��;�$�d▫�▫▫d`▫��;�d�d▫�▫▫d`▫��;�dă▫�▫▫ld`▫{l��{l���c▫��c�#▫cs▫▫�;▫▫▫c��o�▫g'�▫▫▫▫l��▫▫l�dćd$�▫�Č���▫l�▫n�▫���c▫l▫$�{p▫▫���c▫l▫�c▫rl;l{�n��▫���g���
�▫crĄ��ll�
MicroPython v1.17 on 2021-09-02; ESP module with ESP8266
Type "help()" for more information.
>>>
 ets Jan  8 2013,rst cause:4, boot mode:(3,6)

>
```

Depending on your ESP8266 board, this issue will not occur. Anyway, if it happens, the solution is to disable the "*Access Point*" mode. To do this, copy the line below, and paste it into the terminal/console immediately after the ESP8266 board reboots. You will need to watch the reboot timing and enter the command immediately after the reboot. Otherwise, the board will keep rebooting continuously.

Command to be copied and pasted immediately after the board *reboots*:

```
import network; network.WLAN(network.AP_IF).active(False)
```

With MicroPython installed and stable, you can proceed to the next steps.

Note: All activities and commands mentioned in this text will work on ESP8266 and ESP32.

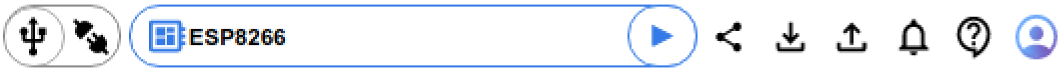
Blinking the on-board LED

The light-emitting diode, or LED, is an indicator light used in many situations to indicate the status of an embedded system. For example, on a modem, it can show whether the Internet is connected. Thus, "*Blink the LED*" is one of the first activities performed to test/learn about a new board/device used to build an embedded system. This section will do a step-by-step activity to flash the onboard LED of the ESP8266 board. In other words, it is not necessary to connect any external LEDs. Just use the one already existent on the board.

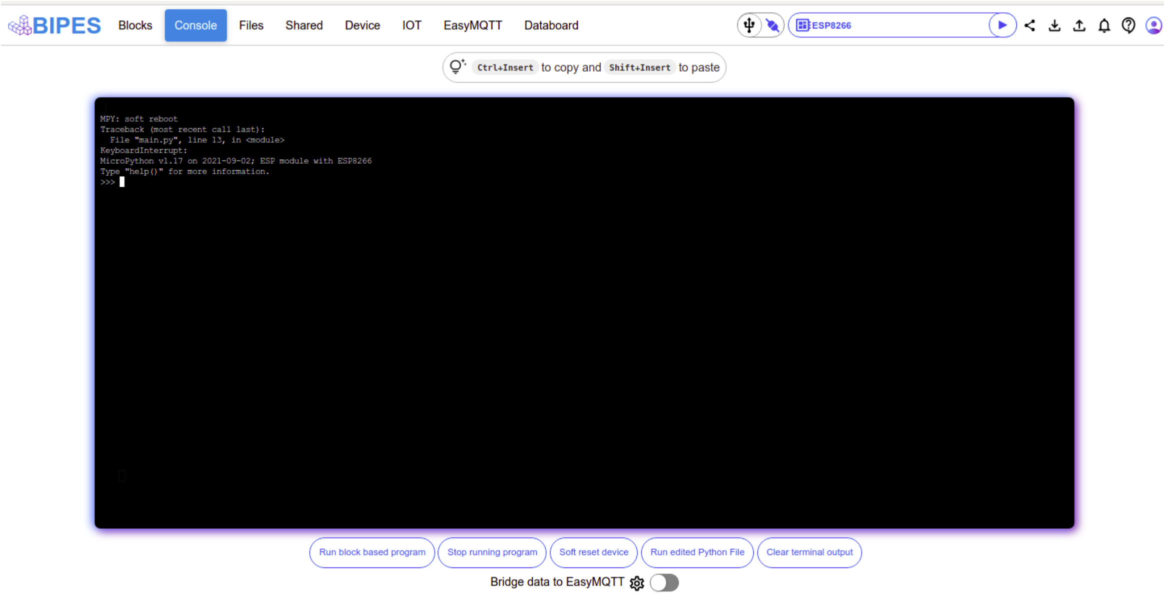
1. Connect the ESP8266 module to the computer's USB port.

2. Access the **BIPES** development environment: https://bipes.net.br/ide/.

3. Choose, in the upper right corner, the **ESP8266** option:



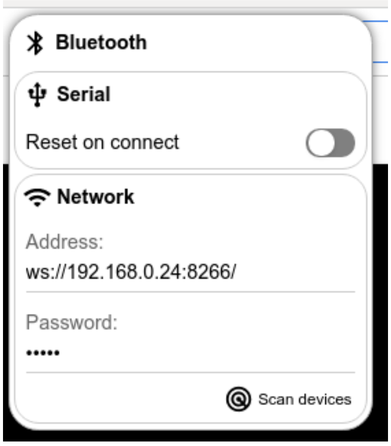
4. Access the **Console** tab:



```
MPY: soft reboot
Traceback (most recent call last):
  File "main.py", line 13, in <module>
KeyboardInterrupt:
MicroPython v1.17 on 2021-09-02; ESP module with ESP8266
Type "help()" for more information.
>>> 
```

5. Click on the connect icon:

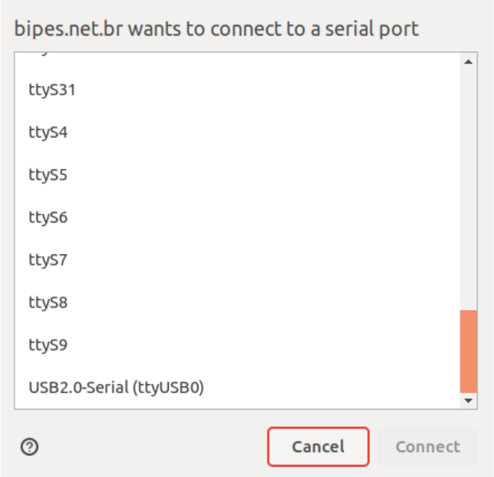
6. Click on **Serial**:



Address:
ws://192.168.0.24:8266/

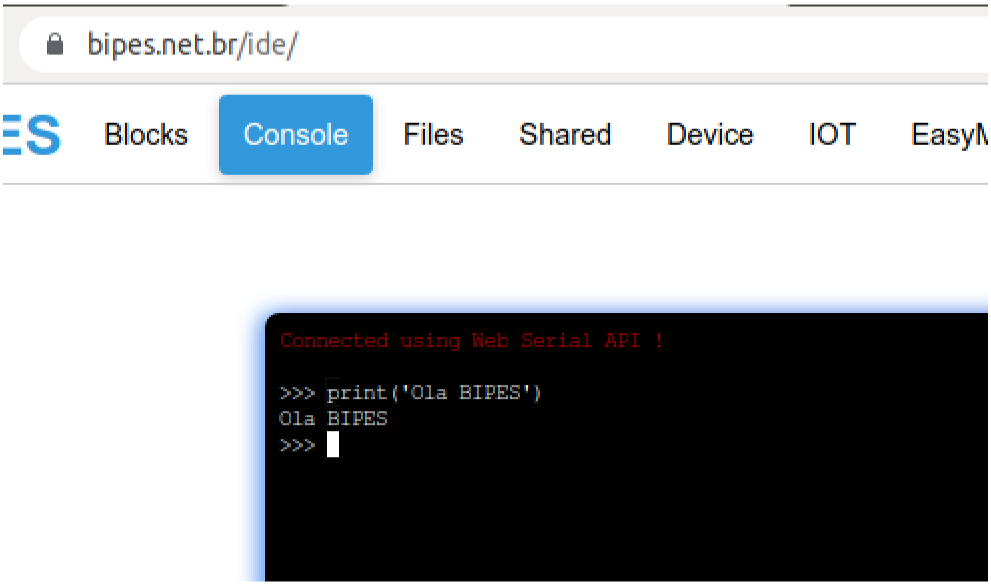
Password:
•••••

7. Choose the port associated with the board and click on **Connect**:


bipes.net.br wants to connect to a serial port

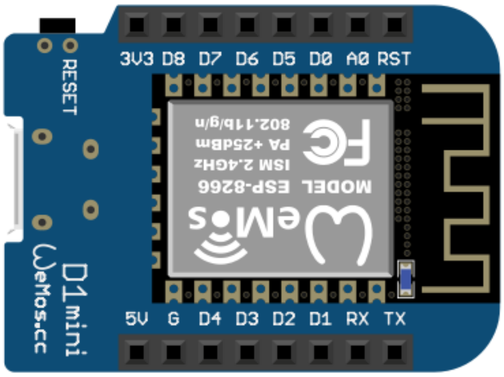
ttyS31
ttyS4
ttyS5
ttyS6
ttyS7
ttyS8
ttyS9
USB2.0-Serial (ttyUSB0)

8. Check if the connection was successful by sending commands to the board. For example, try typing `help()` or `print(“Hello BIPES”)`. If the board is not responding, you can also try resetting the board by pressing the reset button or removing it from the USB port and reconnecting it (return to step 5).

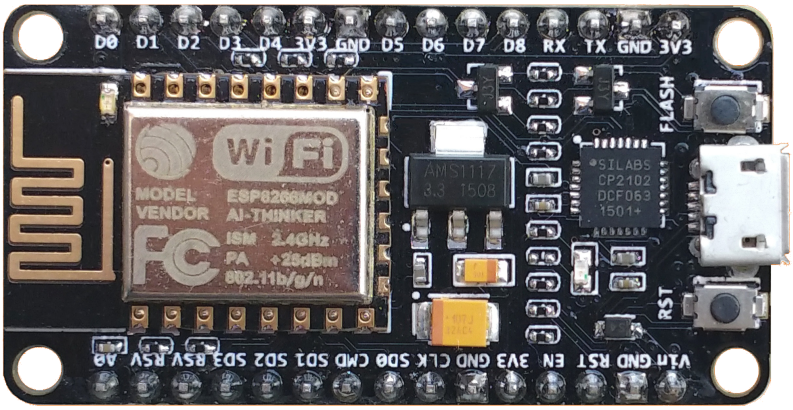


```
Connected using Web Serial API !

>>> print('Ola BIPES')
Ola BIPES
>>> 
```

After connecting, check that each board has a **pin map**. In particular, the board's internal LED indicator light, which we want to use, is connected to pin D4 / GPIO2 of most ESP8266 boards, but check your board pin map to be sure. The figure below shows an example of a WeMos D1 mini board pin map with ESP8266 module:

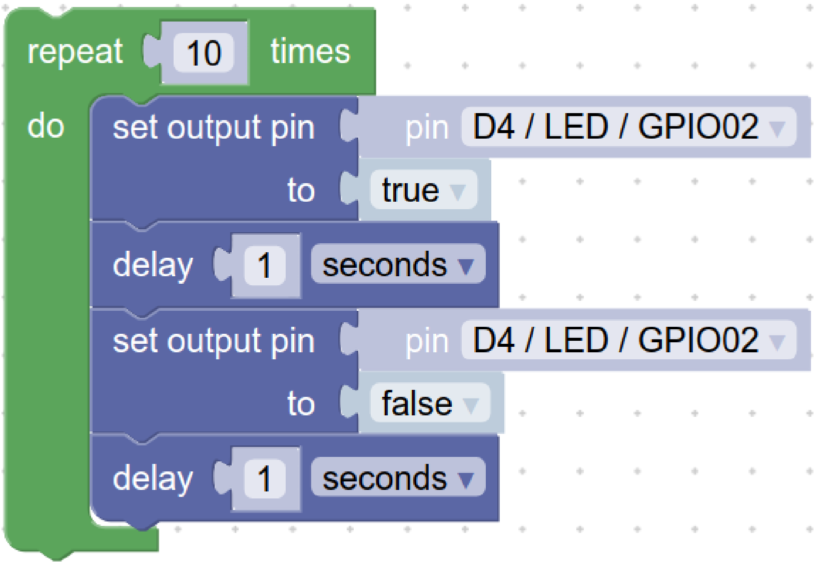


In the image above, note that each terminal to connect external components has a label, such as D8, D7, A0, D2, etc. The 5V pin provides 5 Volts for external circuits, and the 3V3 provides 3.3 Volts. Pin G is the Ground (Ground) of the circuit. Note that there are dozens of models/options of boards with ESP8266 module, so you will need to identify the pins / LEDs, connections of your board. The figure below shows another board option with the ESP8266 module, commonly called NodeMCU or “DevKit” by Chinese suppliers. This board, specifically, has two LEDs: one at GPIO2 and one at GPIO16.


D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
FLASH
RST
Wi Fi
MODEL ESP8266MOD
VENDOR AI-THINKER
ISM 2.4GHz
PA +25dBm
802.11b/g/n
AMS1117 3.3 1508
SILABS CP2102 DCF063 1501+
A0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND Vin

9. Prepare the following program using **BIPES,** as illustrated below.

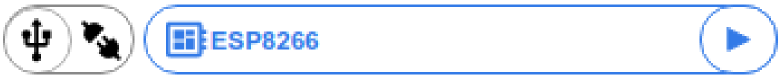
Tip: drag and drop the blocks available in **Loops**; **Timing** and **Machine** → In/Out Pins; Always remember to include a delay so that you can see the LED state-changing between each action. Without delay, the LED will flash so fast that it is impossible to see it blinking with our eyes.



```
repeat 10 times
do  set output pin  pin D4 / LED / GPIO02
                to  true
    delay 1 seconds
    set output pin  pin D4 / LED / GPIO02
                to  false
    delay 1 seconds
```

Note: Due to the internal circuit characteristics of the WeMos D1 mini board, the internal LED of this board turns on with a **low** logic level (0 / false) and goes off with a high logic level (1 / true). This characteristic may vary from board to board.

10. After preparing the program, click on the icon to run it (**Play icon**):

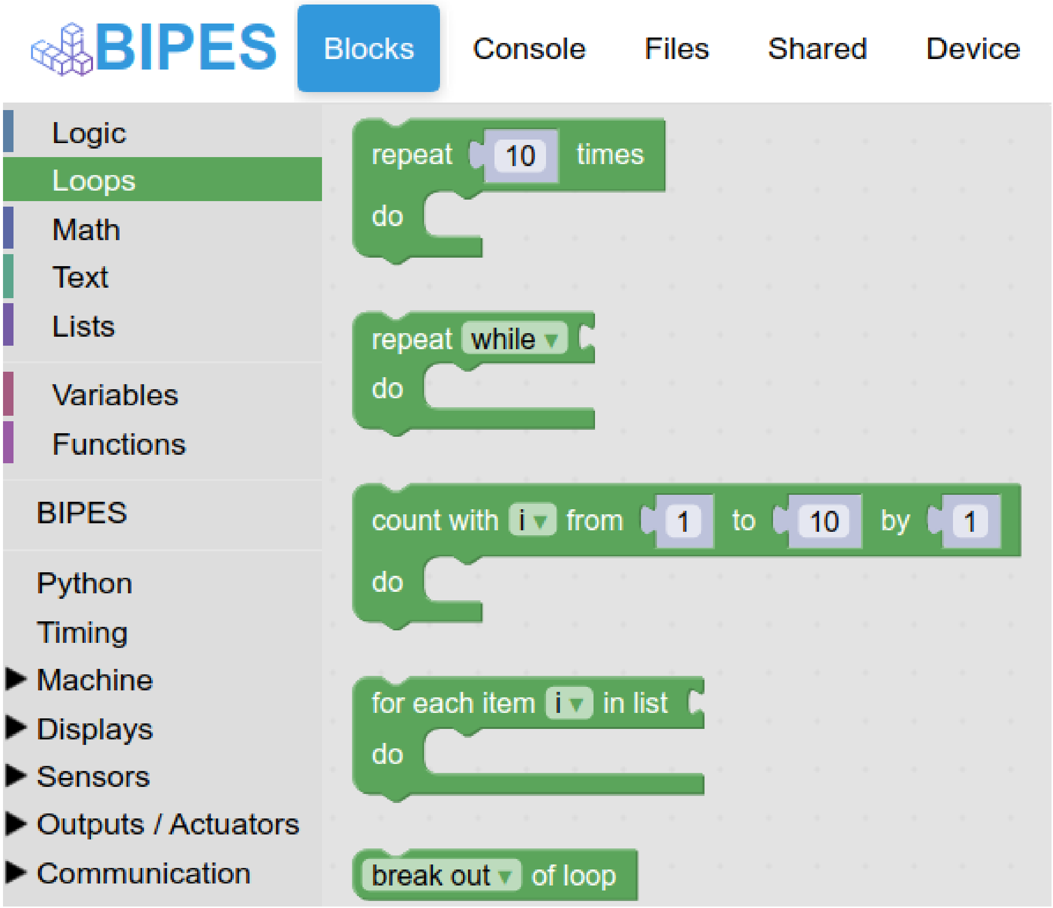


You can check the program being transferred to the board via the **console** if you want. The program will make the LED blink ten times and then terminate.

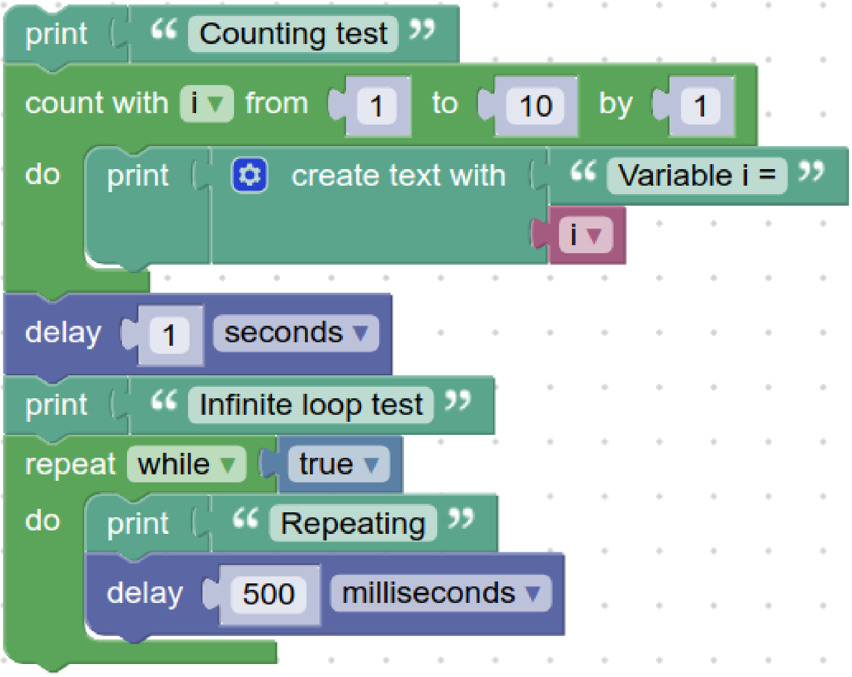
Activity:
Change the program so that the LED flashes with different time intervals and a different number of times.

Tip: Use the user icon in the upper right corner of the screen to choose the language of the BIPES interface (English or other).

We have used the "**repeat 10 times**" block, available in the left toolbox section: **Loops**. However, the same toolbox section also offers other repetition structure options, such as "**count from 1 to 10**" and "**repeat while true**" (repeat infinitely). Indeed, you can click on the numbers and change the values as you want.

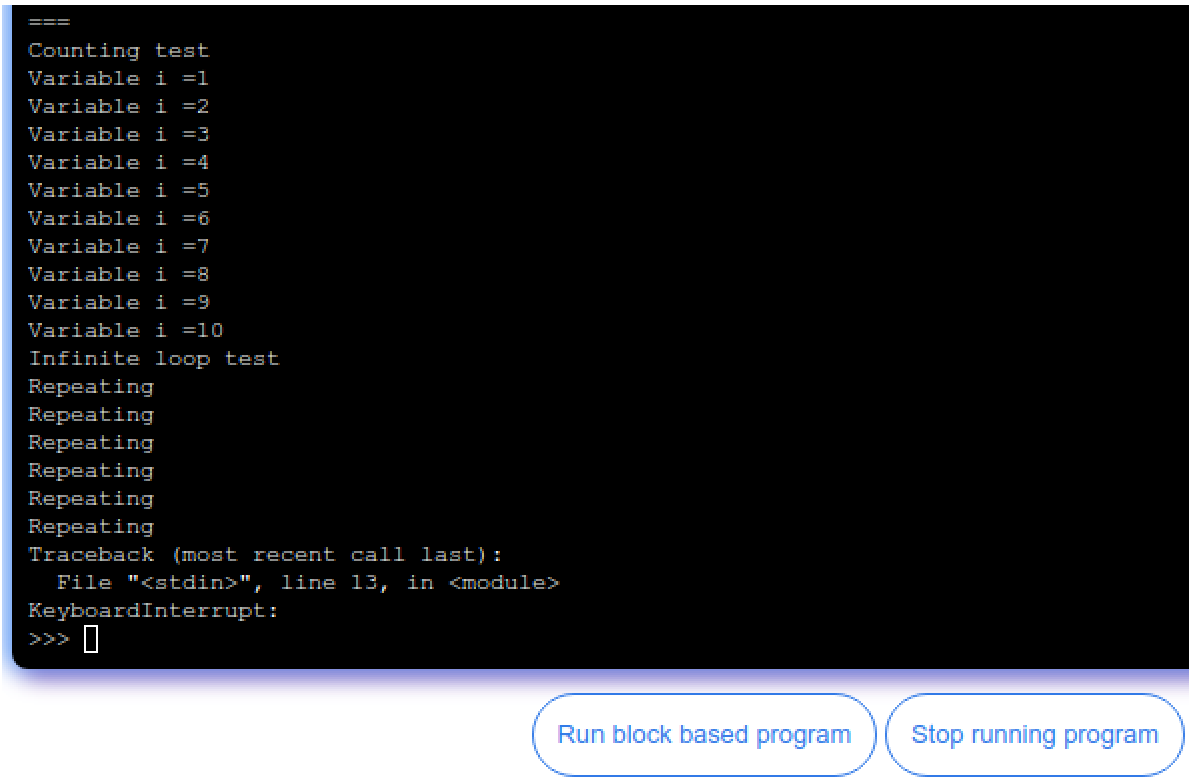


In the case of counting from 1 to 10, each loop iteration updates the "**i**" variable (a storage space for program data), which can be used within the repetition structure. In the example below, the program prints to the **Console** from 1 to 10 with an initial text "**i =** " to show the variable name. Examples of counting and infinite repetition are shown below:


print " Counting test "
count with i from 1 to 10 by 1
do print create text with " Variable i = "
i
delay 1 seconds
print " Infinite loop test "
repeat while true
do print " Repeating "
delay 500 milliseconds

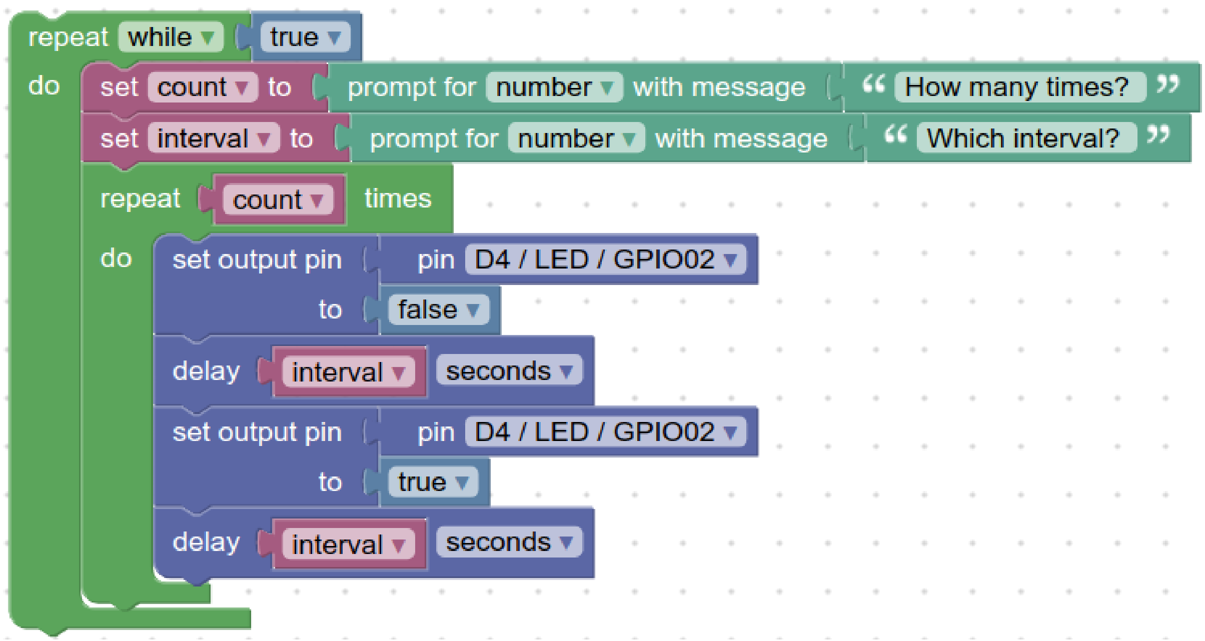
The infinite repetition structure “**repeat while true**” is one of the most used repetition structures in embedded systems since many embedded systems perform a sequence of operations continuously and repetitively while the system is turned on. For example: reading data from a sensor, checking the temperature of an environment, and turning on or off the refrigerator’s compressor, among others. However, it is important to include a method to exit the loop when a condition is met. This can be done with the block “**Loops >> break out of loop**", allowing to "exit" the loop or enter another routine without the need of pressing the reset button directly or by executing CTRL+C via the console. Either way, the CTRL+C command on the console or the “**Stop Running Program**" button will break an infinite loop.

The result can be seen in the **Console** (click Stop to quit the infinite loop).



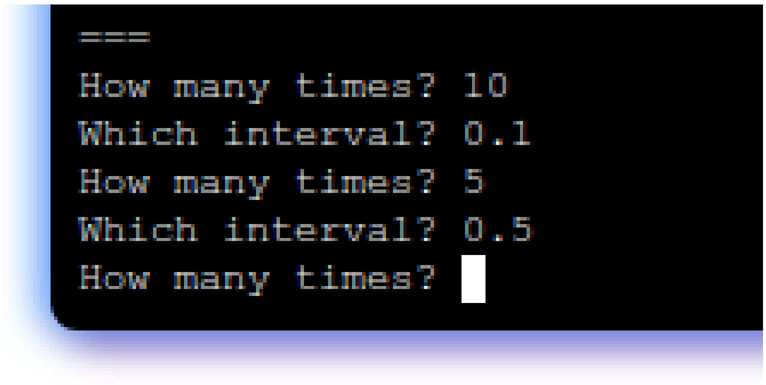
```
===
Counting test
Variable i =1
Variable i =2
Variable i =3
Variable i =4
Variable i =5
Variable i =6
Variable i =7
Variable i =8
Variable i =9
Variable i =10
Infinite loop test
Repeating
Repeating
Repeating
Repeating
Repeating
Repeating
Traceback (most recent call last):
  File "<stdin>", line 13, in <module>
KeyboardInterrupt:
>>> 
```

Another helpful feature associated with the **Console** is requesting user data via the **Console** (prompt the user). The block “**prompt for number**” can be used for this purpose and found in the Toolbar’s text area. Thus, the program below presents an interactive version that flashes the LED. In this version, the program asks how many times the user wants to flash the LED and the time interval with which the LED alternates its state.



```
repeat while true
do  set count to prompt for number with message "How many times?"
    set interval to prompt for number with message "Which interval?"
    repeat count times
    do  set output pin   pin D4 / LED / GPIO02
                     to  false
        delay interval seconds
        set output pin   pin D4 / LED / GPIO02
                     to  true
        delay interval seconds
```

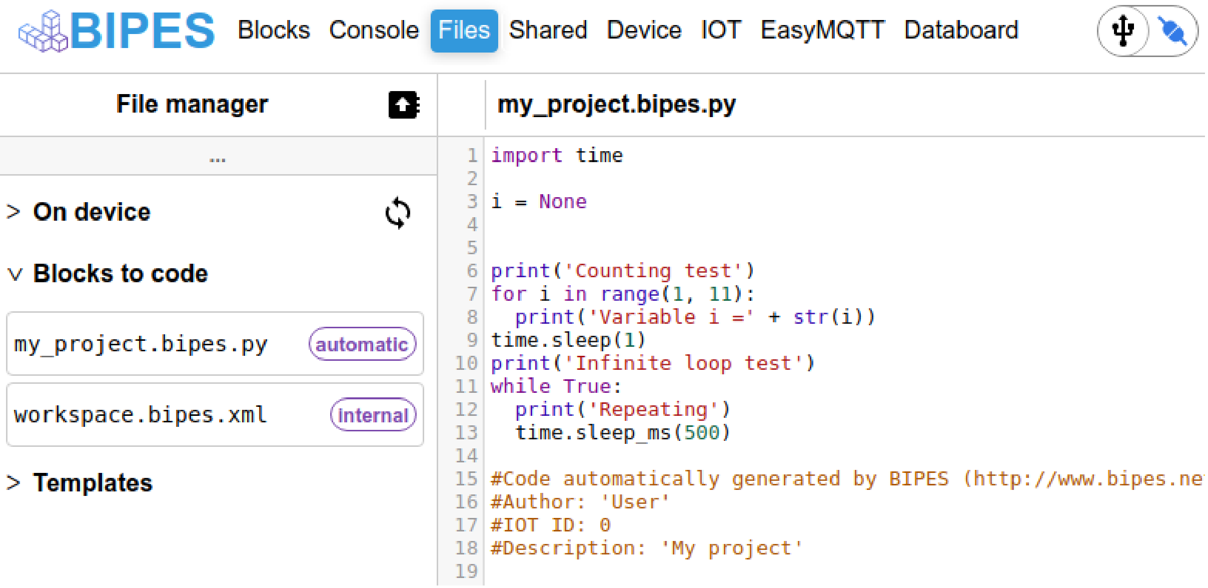
The figure below shows an example of using such a program, where the user typed 10 for the number of times and 0.1 for the time interval. After the program executes the sequence of flashing the LED 10 times, the program asks the user for the options again. For the second time, the user typed 5 and 0.5.



```
===
How many times? 10
Which interval? 0.1
How many times? 5
Which interval? 0.5
How many times? █
```

For the curious:

Check that this block-based program generated source code in Python language (available in the Files tab of BIPES). In that tab, you can see the automatically generated program in the tab **Files >> Blocks to code** area:



```python
import time

i = None


print('Counting test')
for i in range(1, 11):
  print('Variable i =' + str(i))
time.sleep(1)
print('Infinite loop test')
while True:
  print('Repeating')
  time.sleep_ms(500)

#Code automatically generated by BIPES (http://www.bipes.ne
#Author: 'User'
#IOT ID: 0
#Description: 'My project'

```

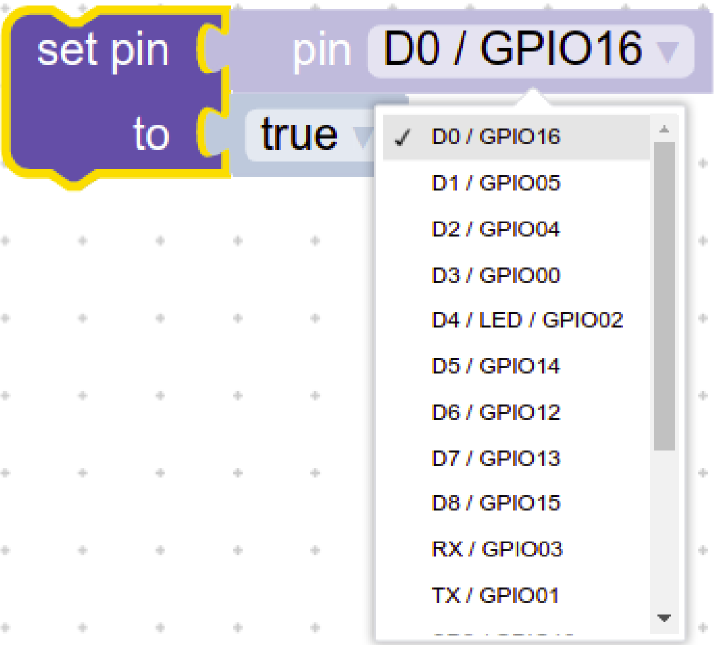
Advanced:

If you want, you can edit the file in Python, save it (button “**Save a copy**”), and run the program edited in text form. As mentioned, the ESP8266 board uses MicroPython, a Python version optimized for embedded systems and microcontrollers. Also, you can save this file as “**main.py**” by the BIPES Files tab (**Save a copy** button).

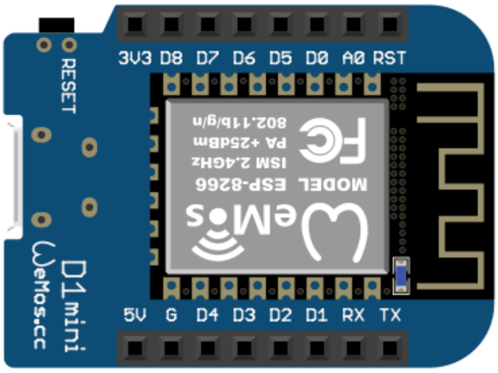
The program “**main.py**” will run automatically every time the board is turned on, autonomously and independently. In other words, your solution will be saved and embedded in the board and will not depend on BIPES or on the computer to work! If you want to test it, you can connect the board to a power bank or a power adapter plugged on the mains and verify that the program/system works autonomously.

Digital input and checking a condition

Microcontrollers, in general, have two main types of inputs: digital and analog. Inputs are usually associated with *General Purpose Input Output* (GPIO) pins denoted on the board as GPIOX, where X is the pin number with codes such as D0 or D1. Therefore, when choosing the board model in BIPES, the pins of this board will be automatically listed for choice, facilitating programming, as shown in the figure below:



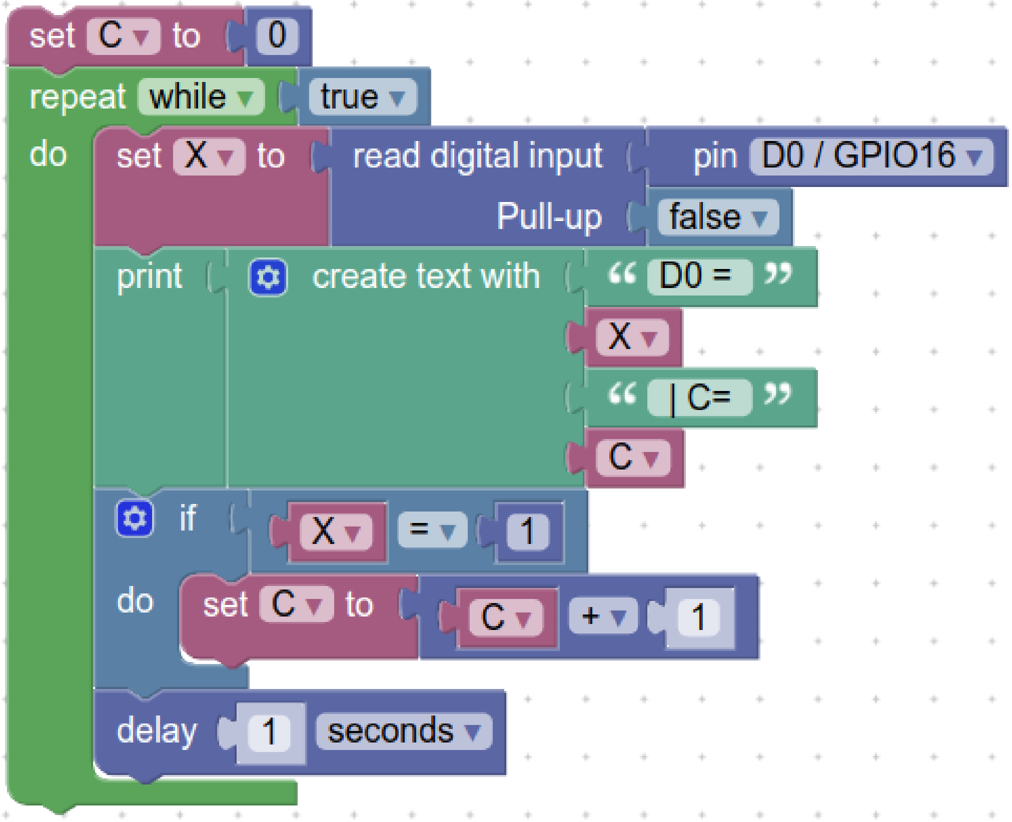
Note that this list of pins will be associated with the physical pins of the board:



In the case of digital mode, these pins can be used to control output devices, and can assume two states: [on / true / **1**] or [off / false / **0**]. Thus, different devices can be connected and controlled by these pins. The same pins can also be used in input mode, in order to “read” digital input signals: [on / true / **1**] or [off / false / **0**].

Since the ESP8266 module works with 3.3 Volts, a potential difference of 3.3V between the referred pin and the reference (G) represents logic **1**, while a zero potential difference (0V) represents logic **0**.

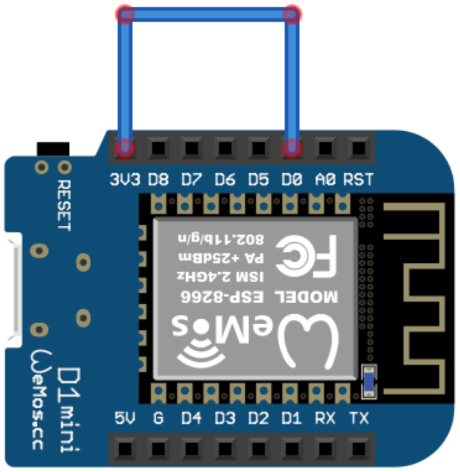
The example below shows a program that monitors the digital input and checks for a condition (**IF**) so that a variable (program data storage space) has its value incremented by one unit every second **IF** the physical status of this pin is 1 (active). The program starts by setting the value of variable C, a temporary counter, to 0 [**set C to 0**]. The program then proceeds to an **infinite loop [repeat while true]**. Within this repetition structure, variable X receives the reading from digital pin D0 [**set X to read digital pin D0**], so that variable X will receive the value 0 if pin D0 has 0 Volts or 1 if pin D0 has 3.3 Volts. Next, the program prints the value of variable X and variable C on the **Console**, in the format "D0 = n | C = n", via the command **print.** Then, the instruction **IF** checks if the variable X, which has the digital pin reading, is equal to 1, and if so, it increments the value of C by 1. Finally, the program waits 1 second and restarts the sequence.

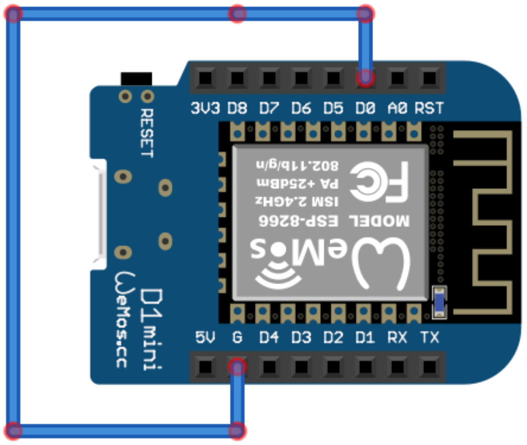


```
set C to 0
repeat while true
do  set X to read digital input  pin D0 / GPIO16
                                 Pull-up false
    print create text with " D0 = "
                           X
                           " | C= "
                           C
    if X = 1
    do  set C to C + 1
    delay 1 seconds
```

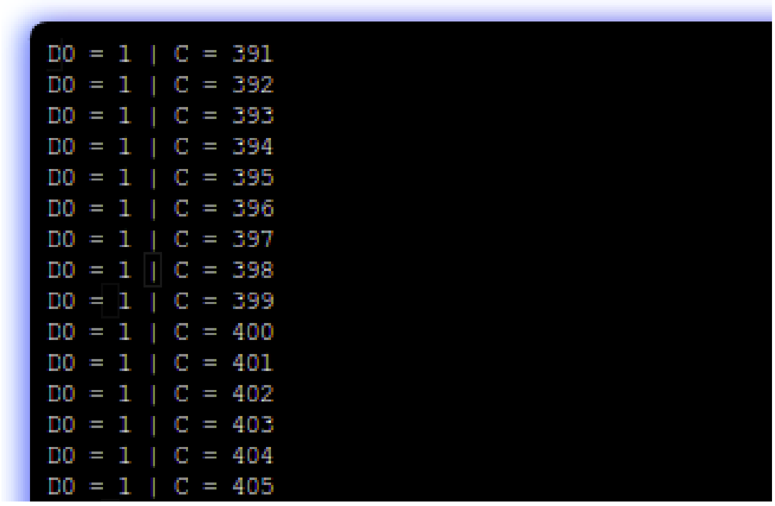
⚙	Check that the block "**create text with**" has a small blue gear. It is possible to configure this block's number of inputs by clicking on this icon. Over the following few practices, use this blue button to configure block options whenever necessary.

The following figure shows a simple way to test the program we just created. First, start the program execution and use a *jumper cable* to switch the input value of the pin configured in the program (D0), by connecting it to pin 3V3 (logic level 1) and later to pin G (ground, logic level 0). Notice the changes on the **Console** tab and the counter increasing every second when the received logic level is 1.





Example of result that can be seen in **Console**:

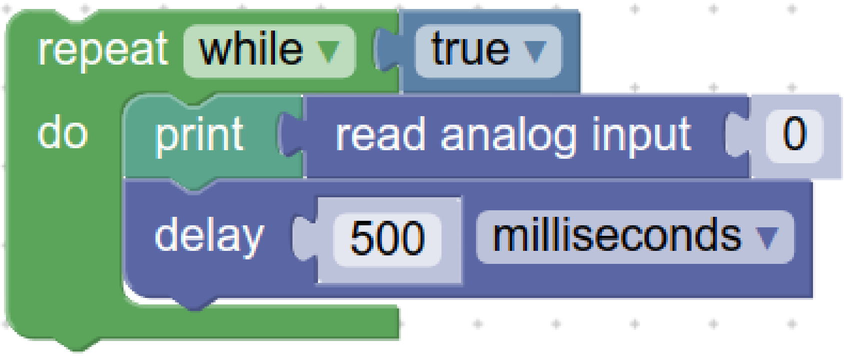


```
D0 = 1 | C = 391
D0 = 1 | C = 392
D0 = 1 | C = 393
D0 = 1 | C = 394
D0 = 1 | C = 395
D0 = 1 | C = 396
D0 = 1 | C = 397
D0 = 1 | C = 398
D0 = 1 | C = 399
D0 = 1 | C = 400
D0 = 1 | C = 401
D0 = 1 | C = 402
D0 = 1 | C = 403
D0 = 1 | C = 404
D0 = 1 | C = 405
```

Based on this example, several applications can be created, such as counting the number of vehicles or people that have passed through a portal, checking whether a door is opened or closed, checking the presence of people by presence/alarm sensor, checking the level of a water tank (complete or not) with a buoy-type digital sensor, among other possibilities.

Analog input and LDR light sensor

Another possibility of reading external inputs is through analog input pins. While digital pins only detect signals **0** or **1**, analog pins are sensitive to variable measurement values within a predefined range. For example, they can measure voltage, temperature, or other values that vary between various possibilities. The ESP8266 has only one analog input port (pin A0), which can measure input values between 0 and 3.3 Volts, internally mapped between 0 and 1023, with 0 being equivalent to 0 Volts and 1023 equivalent 3.3 Volts[1]. The ESP32, in turn, has several analog input pins, which can be used simultaneously. To use the analog input from the ESP8266 module, you can use the following program with the block "**Read ADC Input" or** "**read analog input"**, which reads the analog pin and is available in the section **Machine** → **In/Out Pins** of BIPES toolbox.



```
repeat while true
do  print read analog input 0
    delay 500 milliseconds
```

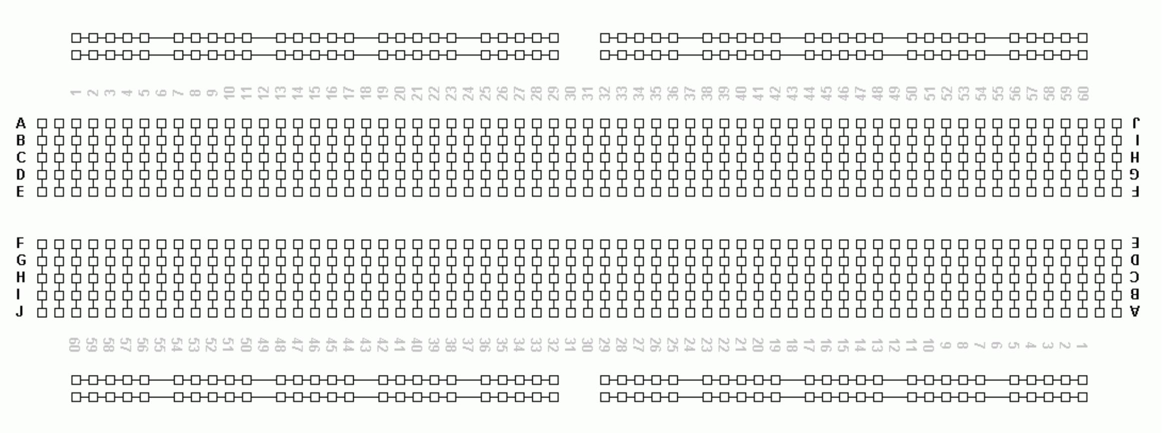
You can see values ranging from 0 to 1023 on the **Console tab** while running the above program. During the test, use a *jumper cable* to connect pin A0 to G and observe the reading of 0. Then, connect pin A0 to pin 3V3 and note a value close to 1023 is printed on the Console. Attention: ESP8266 module input pins support a maximum of 3.3 Volts. Do not apply higher electrical voltage values at the risk of damaging or burning the module.

Activity:
Considering that the reading 1023 corresponds to 3.3 Volts, modify the above program so that it shows, on the **Console** tab, the value in Volts applied to pin A0 of the module.

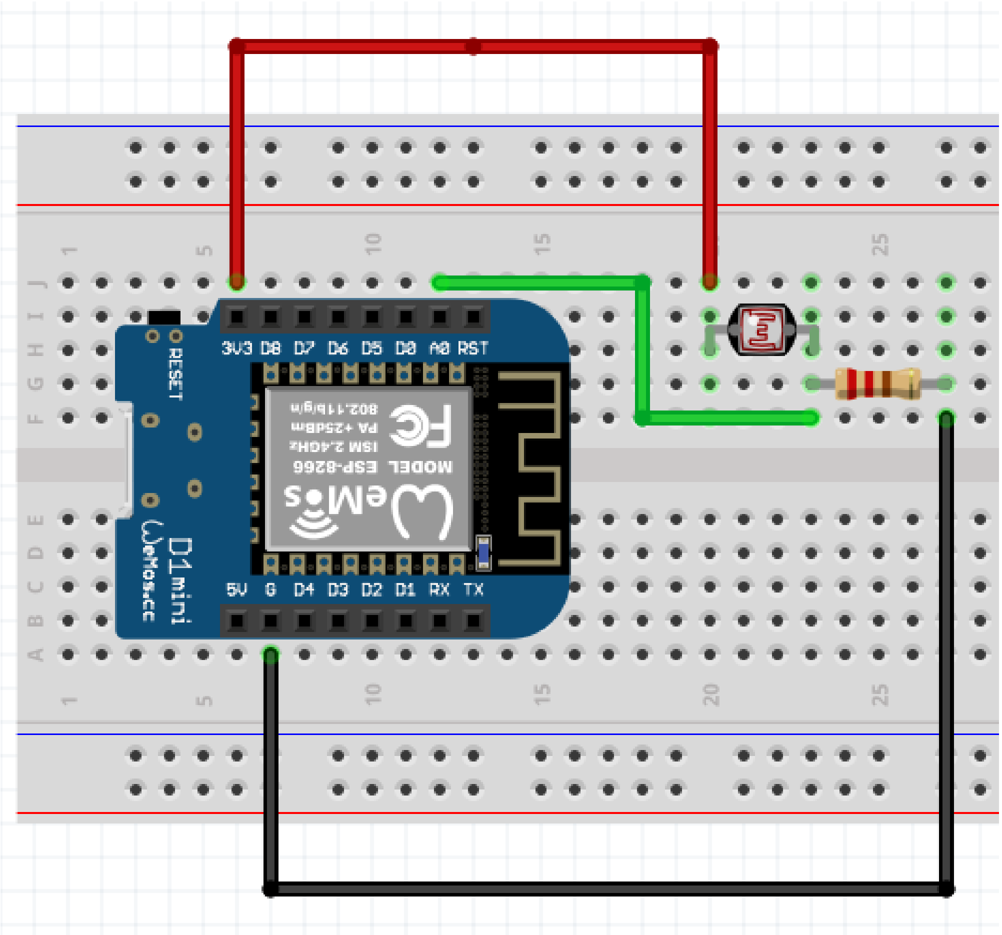
We will now use an LDR (*Light Dependent Resistor*) sensor, which varies its electrical resistance according to the ambient light and can be monitored through an analog input of the ESP8266 module. First, as shown in the following figure, assemble the circuit using a prototyping board (*protoboard or breadboard*). The figure below illustrates how the *breadboard* interconnects parts: columns at its center and rows at its borders. Thus, all components connected in the same column will automatically have their terminals electrically interconnected.

---

[1] The block **read analog input** converts the value of the analog voltage signal at the input to a 10-bit digital value.



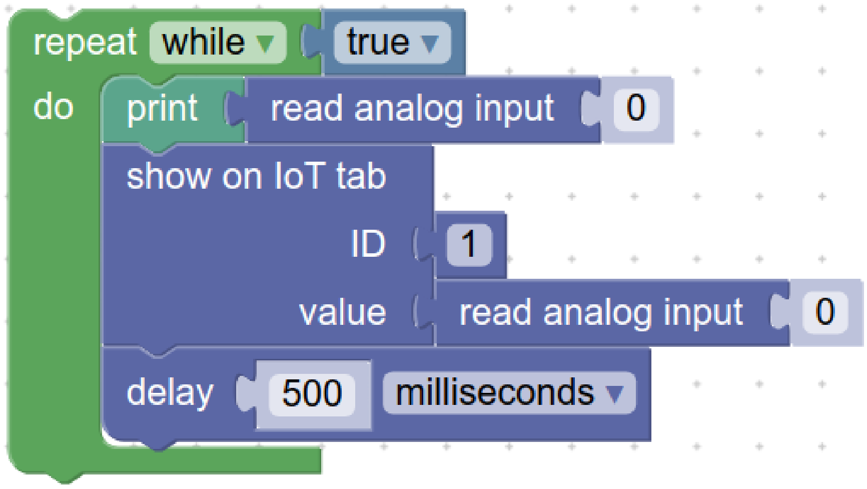
Circuit assembly with the ESP8266 module, LDR light sensor, and a 220-ohm resistor (red, red, brown):



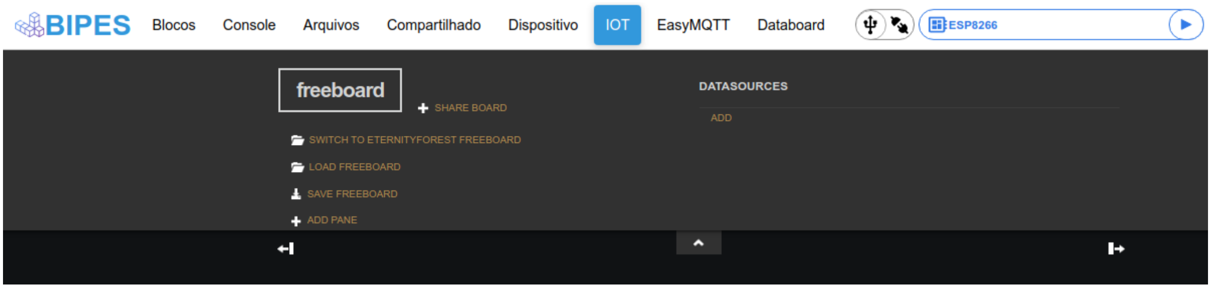
The setup above is based on an electronic configuration called voltage divider so that the proper selection of 2 resistors connected in series can set the output voltage value at the common terminal between the two resistors. In the case of this circuit, the LDR sensor is a variable resistor, and as the lighting changes, the resistance of the LDR also changes, changing the output voltage, which is measured by pin A0 (green wire). The red cable (V) is connected to the 3V3 of the ESP8266 board supplying current to the circuit. The output cable (green) is connected to the board's analog input (AIN / ADC0), and the black wire is connected to the board's ground (G or GND), closing the circuit. Now we can use the already created program to analyze the lighting variation detected by the LDR.

The above program will "read" the analog input and show it in the BIPES **Console** tab. Try to cover the LDR sensor with your hand and see the values changing on the console. Try also to point a flashlight to the LDR and check the readings.

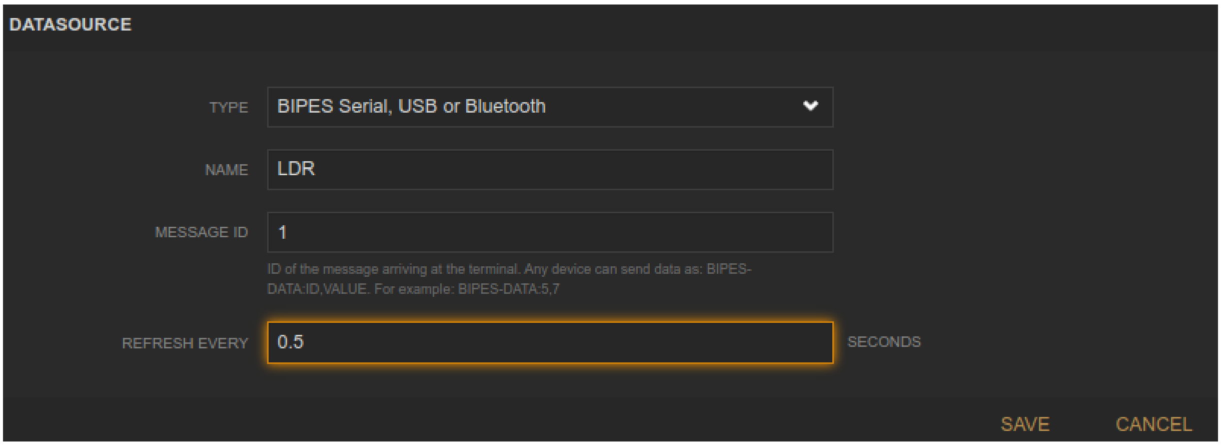
Next, we can go further with this setup and visualize the readings graphically. To do this, add the block "**show on IoT tab"**:



Now use the BIPES **IOT** tab:



In **DATASOURCES**, click **ADD** and choose "**BIPES Serial, USB or Bluetooth**" and type **LDR** as the name, **1** as the message ID, and **0.5** as the refresh rate:


DATASOURCE

TYPE: BIPES Serial, USB or Bluetooth

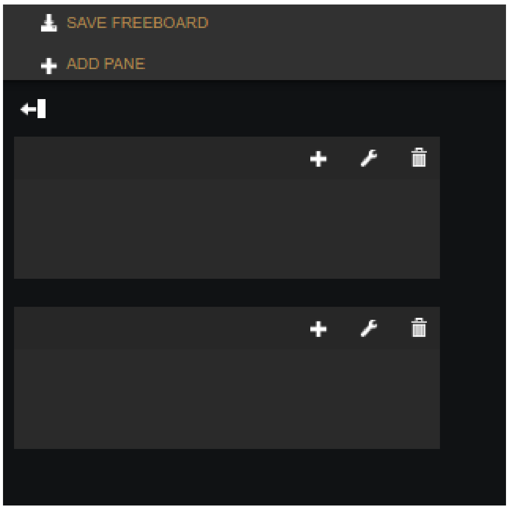
NAME: LDR

MESSAGE ID: 1

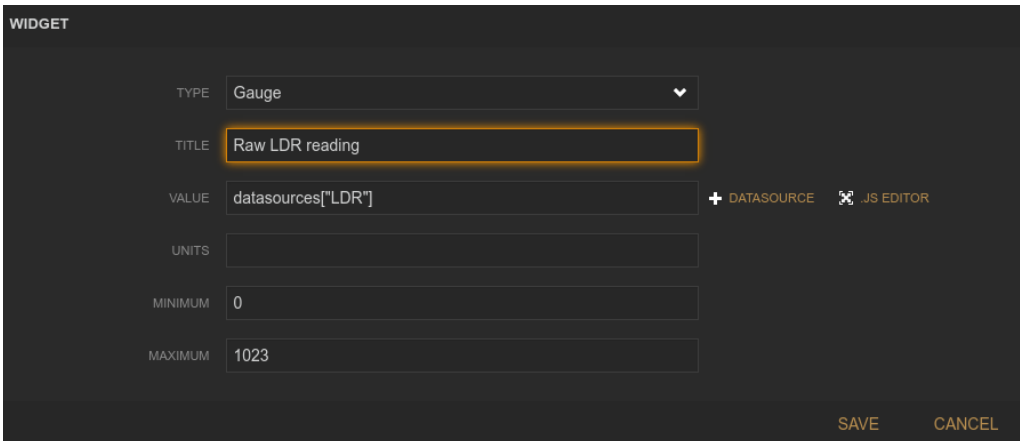
ID of the message arriving at the terminal. Any device can send data as: BIPES-DATA:ID,VALUE. For example: BIPES-DATA:5,7

REFRESH EVERY: 0.5 SECONDS

Save and add one or more “***pane***” (*dashboard* panels*)*, with the button **ADD PANE**:



Drag and drop the “*panes*” to adjust their positions at your preference. Then, in each “*pane*”, add visualization components (by clicking the icon **+**), as you prefer, and enter “*datasources[“LDR”]*” in the field **VALUE** of each widget you added.


WIDGET

TYPE: Gauge

TITLE: Raw LDR reading

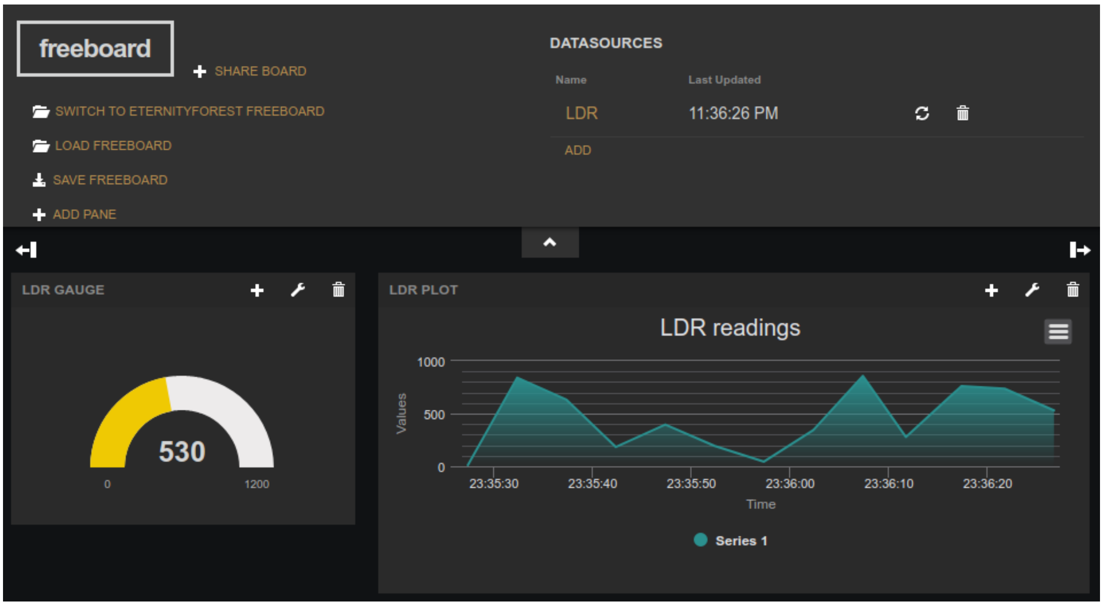
VALUE: datasources["LDR"]

UNITS:

MINIMUM: 0

MAXIMUM: 1023

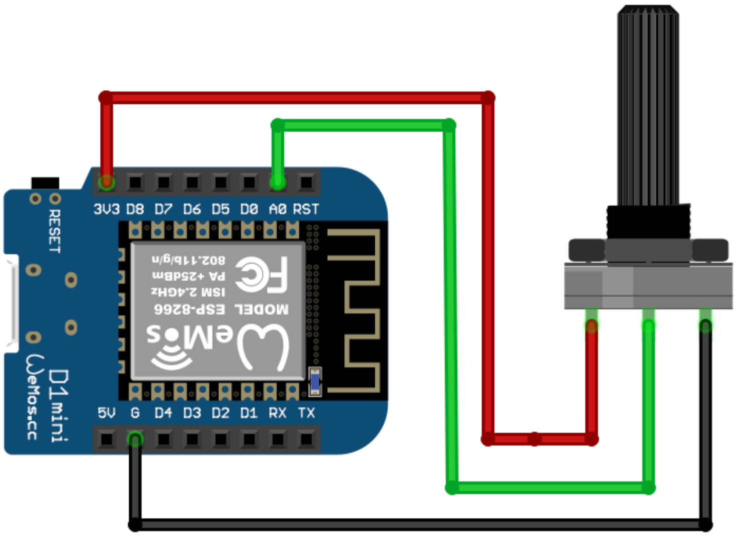
In the example below, we selected a widget of type ***Gauge*** and one of the type ***Time series (Highcharts)***:



Run the program and check the visualization. By varying the lighting that the LDR sensor is exposed to, the readings will change and be displayed in real-time on the widgets you have added to the dashboard.

Activity:
Swap the LDR for a potentiometer and check the values changing as you rotate the potentiometer shaft. Discuss / research what is / how the potentiometer works.

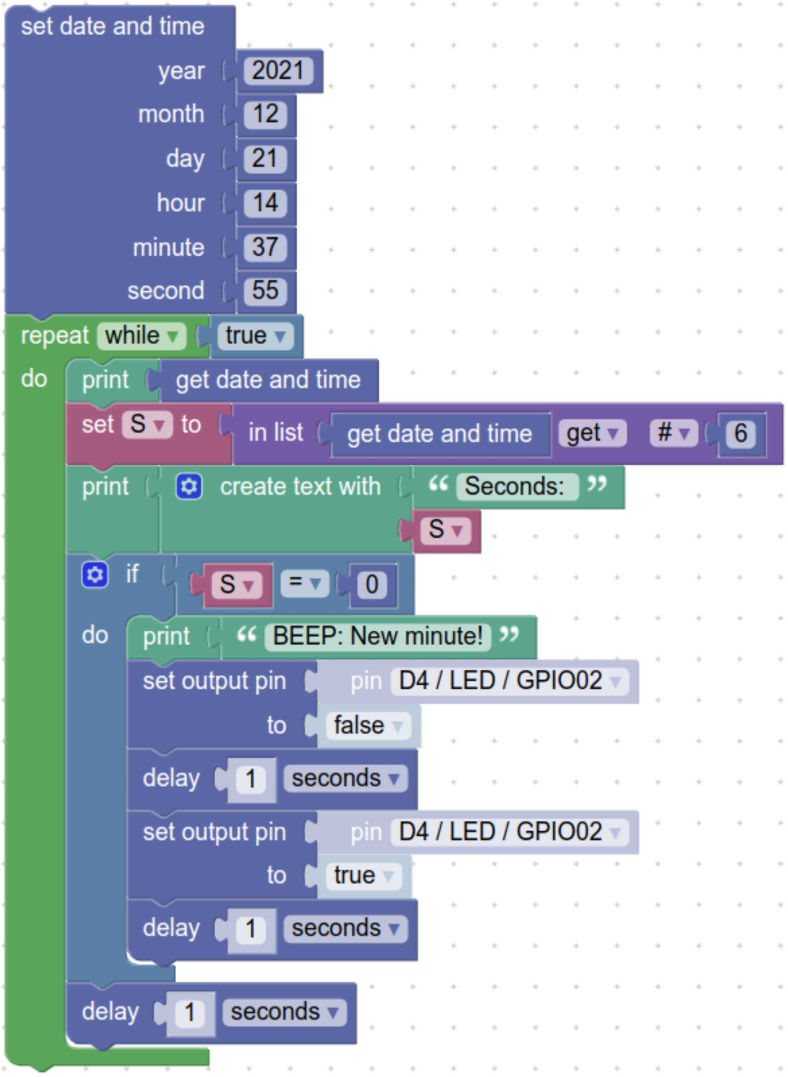
Circuit assembly suggestion:



Date and time (RTC)

The MicroPython installed on the ESP8266 or ESP32 board makes it possible to access a *Real-Time Clock* (RTC) feature. RTC allows the system to set a date and a time and keep an up-to-date record of the time, independently of the programs executed on the board. The system's internal RTC can be adjusted in several ways: manually, from an external RTC with battery, which provides the correct date and time, even if the system is turned off, or via the Internet, using the block ***Network Time Protocol*** **(NTP)**.

The following example shows a program that starts the clock with the date/time 12/21/2021 at 14:37:55. Next, the program treats the date and time as a list structure, separated by commas: **(2021, 12, 21, 1, 14, 37, 55, 0)**[2]**.** It is then possible to use the block "**in list**" to take the sixth element (starting from 0), which corresponds to seconds, and store its value in the variable S. Then, it is verified **IF** the variable S is equal to 0, and when it does, the LED is turned on for one second.



```
set date and time
    year 2021
    month 12
    day 21
    hour 14
    minute 37
    second 55
repeat while true
do  print get date and time
    set S to in list get date and time get # 6
    print create text with "Seconds: "
                           S
    if S = 0
    do  print "BEEP: New minute!"
        set output pin pin D4 / LED / GPIO02
                    to false
        delay 1 seconds
        set output pin pin D4 / LED / GPIO02
                    to true
        delay 1 seconds
    delay 1 seconds
```

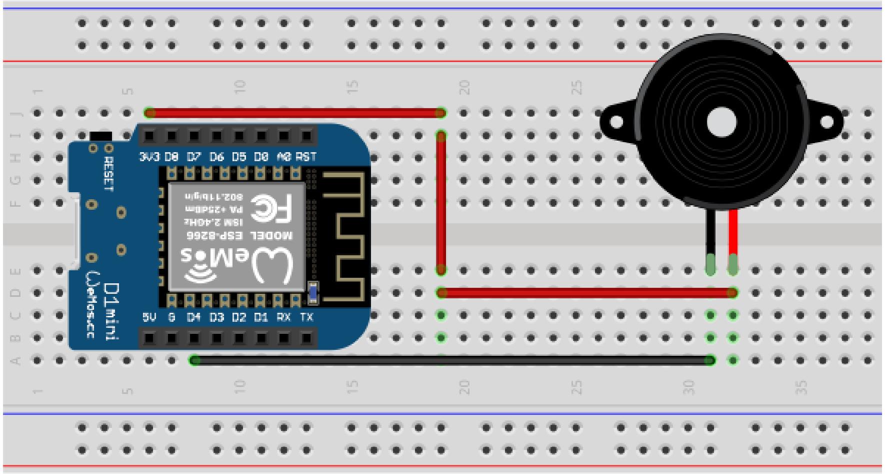
---

[2] (year, month, day, day of week, hour, minute, second, millisecond)

The result can be seen in the **Console**:

```
===
(2021, 12, 21, 1, 14, 37, 55, 222)
Seconds: 55
(2021, 12, 21, 1, 14, 37 , 56, 226)
Seconds: 56
(2021, 12, 21, 1, 14, 37, 57, 231)
Seconds: 57
(2021, 12, 21, 1, 14, 37, 58, 236)
Seconds: 58
(2021 , 12, 21, 1, 14, 37, 59, 241)
Seconds: 59
(2021, 12, 21, 1, 14, 38, 0, 245)
Seconds: 0
BEEP: New minute!
(2021, 12, 21, 1, 14, 38, 3, 233)
Seconds: 3
(2021, 12, 21, 1, 14, 38, 4, 232)
Seconds: 4
(2021, 12, 21, 1, 14 , 38, 5, 227)
Seconds: 5
```

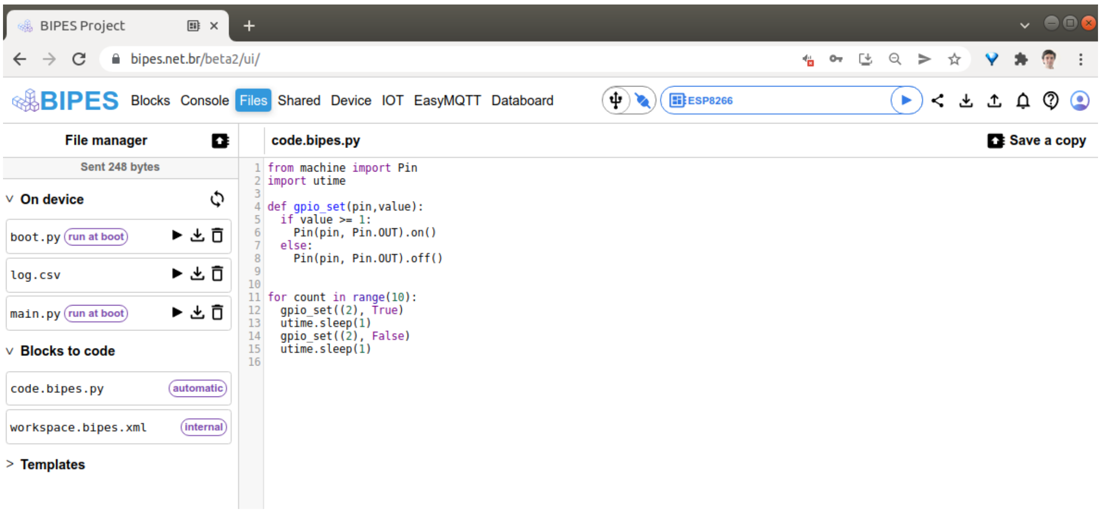
A *buzzer* can be connected between pin 3V3 and the digital pin D4, allowing the system to emit a 1-second beep every minute. For example, the figure below shows the possibility of connecting a *buzzer* using pins 3V3 connected to the + of the *buzzer* and the D4 to the other buzzer terminal*.* *The Buzzer* circuit is shown in the following figure.



Note that the date and time are not lost even with the **delay** of 3 seconds (2 seconds inside the IF and one on the repeat loop). The reason is that the RTC maintains an up-to-date record of the date and time, regardless of the program's operation. The current date and time will be obtained whenever the block "**get date and time"** is used. Later, you can use an external RTC, which, backed up by a battery, will keep the date and time always up to date, or get the date and time from the Internet via the NTP block.

Files on the board

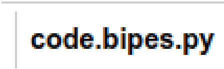
If you have already used Arduino, you must remember that in Arduino, there is no concept of files stored on the board. In the case of BIPES with MicroPython, each board can store several files, which can be programs, data, configurations, or any other information you want to record in files. Therefore, file management in BIPES is quite simple and can be done through the **FILES** tab. This entire file management environment can be used via USB cable or WiFi if the board is connected to a WiFi network. The following figure shows an example of the files tab **BIPES**:



```python
from machine import Pin
import utime

def gpio_set(pin,value):
  if value >= 1:
    Pin(pin, Pin.OUT).on()
  else:
    Pin(pin, Pin.OUT).off()


for count in range(10):
  gpio_set((2), True)
  utime.sleep(1)
  gpio_set((2), False)
  utime.sleep(1)

```

The table below explains each button available on the **FILES** tab:

⬆	**Upload script to the device** Sends a file from your computer to the board
⬆ **Save a copy**	**Save a copy** Saves a copy of the opened file with a custom name on the board
↻	**Refresh device file list** Updates the file list
▶	**Run file** Runs a Python program stored in that file. It is also possible to use keyboard hotkeys: Ctrl+Shift+R to start a program and e Ctrl+Shift+S to stop a program. Hotkeys can be used from any BIPES tab.
⤓	**Download file** Download the file on the board to your computer
🗑	**Delete file** Deletes the file from the board

 code.bipes.py	**Filename** Name of the last file opened on BIPES text editor or to be saved by the command **Save a copy**. You can click on this area and edit the file name. Thus it is possible to save copies or save modifications.

In the area **Blocks to Code**, you can see the program automatically generated from the BIPES blocks, which can be edited and have copies saved. In addition to storing programs and settings, it is also possible to use files to store collected data without the need for an Internet connection. For example, the program below reads the analog input and writes it in the **log.csv** file associated with the reading date and time, with data collection every 60 seconds. Such a program could be used for some environmental data collection systems.

The collected data is stored in the board's FLASH memory and kept even if there is a power failure. By the way, MicroPython is also stored on the FLASH memory, so it manages this memory to accommodate the system and user files and programs.

When the board is connected to a computer, it is possible to download the log.csv file, which will have the following format:

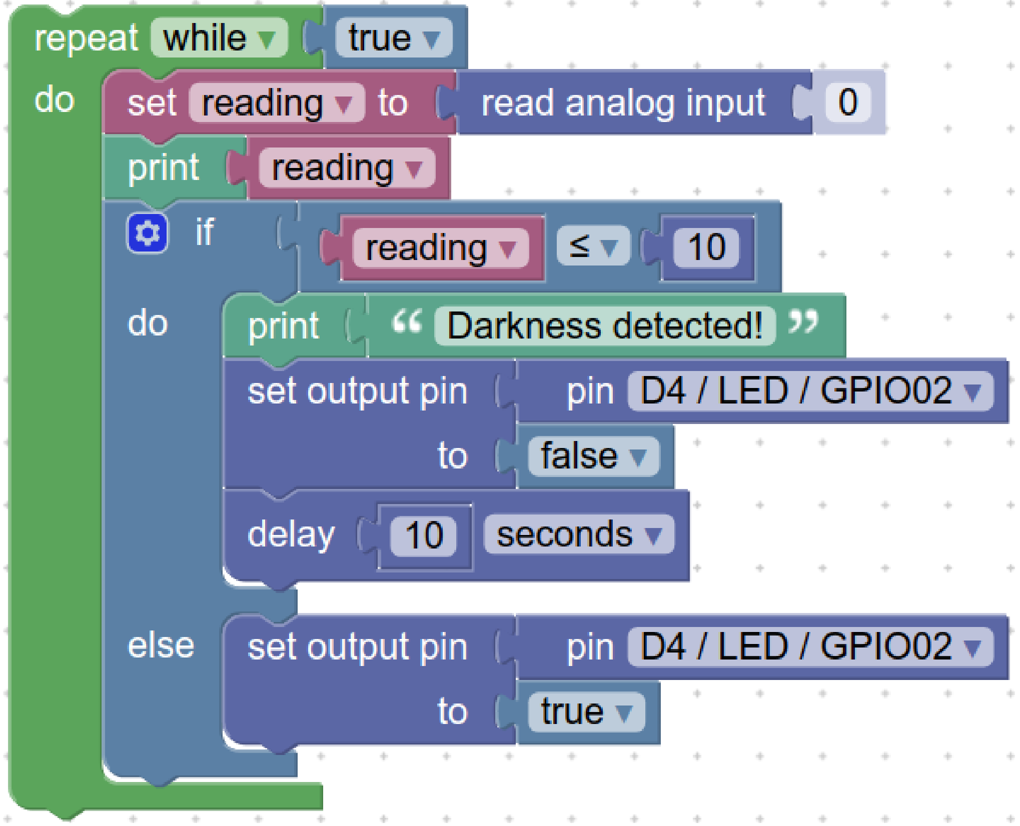
```
(2021, 12, 15, 2, 22, 42, 50, 3):8
(2021, 12, 15, 2, 22, 43, 0, 60):8
(2021, 12, 15, 2, 22, 43, 10, 119):8
(2021, 12, 15, 2, 22, 43, 20, 173) :8
(2021, 12, 15, 2, 22, 43, 30, 228):8
```

As you can see, the date and time are written on each line of the file, and then the analog input reading is also written after the colon “:”, as instructed in the program by the block **set L to create text with M : read analog input 0**. If necessary, the MicroPython also allows these operations to be carried out on an SD Card memory card, allowing greater storage space for data collection. SD Cards can be easily added to projects using boards with built-in slots for SD Cards or SD Cards shields.

Finally, after reading the last section of this book (sending data to the Internet), you could create programs that collect data to files and send these data to the Internet when there is a connection available, avoiding any data loss, as the data file can be stored on the FLASH memory or SD Card.

Checking a Condition (Part 2)

We have already used the conditional structure **IF** to take actions based on a digital input. Now, we discuss another function of an embedded system, which is to make a decision based on the reading of an analog sensor—for example, activating a device when a sensor detects some predetermined level. In the case of the circuit with the LDR, an example would be to activate the LED for 10 seconds when the sensor detects that it has gone dark. For this example, we define “dark” when the LDR provides a measurement value less than 10 (within the range of 0 to 1023). For example:



```
repeat while true
do  set reading to read analog input 0
    print reading
    if reading ≤ 10
    do  print "Darkness detected!"
        set output pin  pin D4 / LED / GPIO02
                    to  false
        delay 10 seconds
    else
        set output pin  pin D4 / LED / GPIO02
                    to  true
```

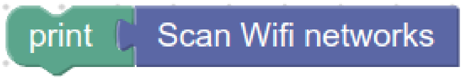
Test the program and try to adjust different levels for darkness detection.

Activity:
Make the LED blink while the lighting is below the specified level, and stop blinking when it is above this level.

Also, save the program as main.py using the **FILES** tab and test the system disconnected from the computer, for example, connected to a Power Bank or USB power adapter connected to the mains. Note that the program runs standalone on the ESP8266 and not on your computer!

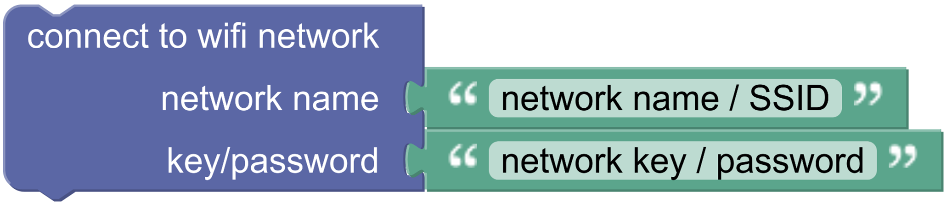
Networking: List Wi-Fi networks

In the following activities, we will connect our board to the Internet! But before that, it is possible to list available networks. Try it yourself. The network options are at the end of the toolbox, under the “**Network and Internet**” option. For example, the following program lists the available wifi networks:



print Scan Wifi networks

Networking: Connecting to the Internet



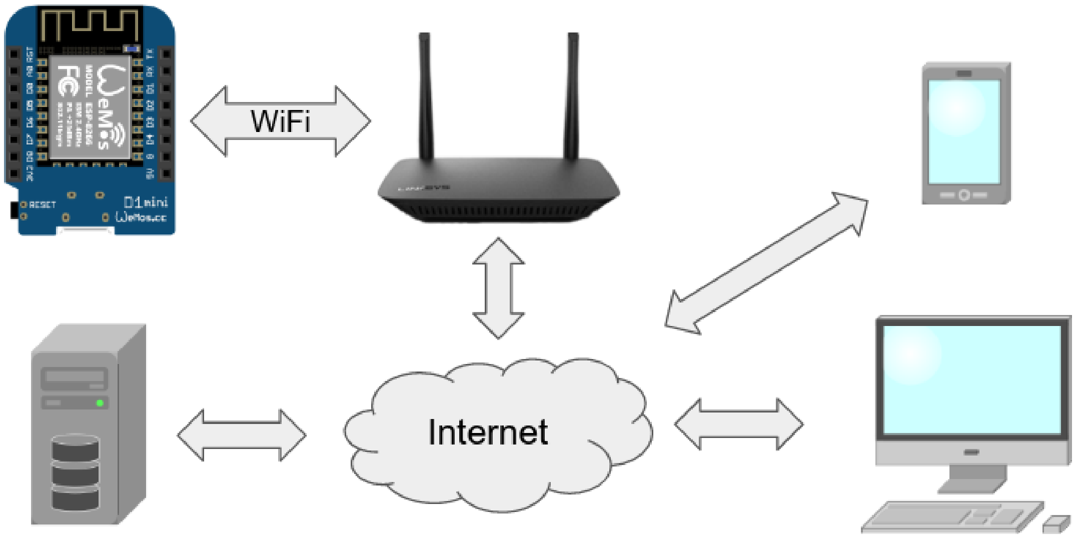
connect to wifi network
network name “ network name / SSID ”
key/password “ network key / password ”

Use the block above to inform the network name and password and connect to the Internet. Next, try running the program so that the board connects to the Internet. This block will automatically obtain an IP address from the network using Dynamic Host Configuration Protocol (DHCP). Eventually, you might need to know the IP obtained: such information can be obtained using the block **Wifi current IP**. After listing the networks and connecting to the Internet, we can now send data to the cloud!

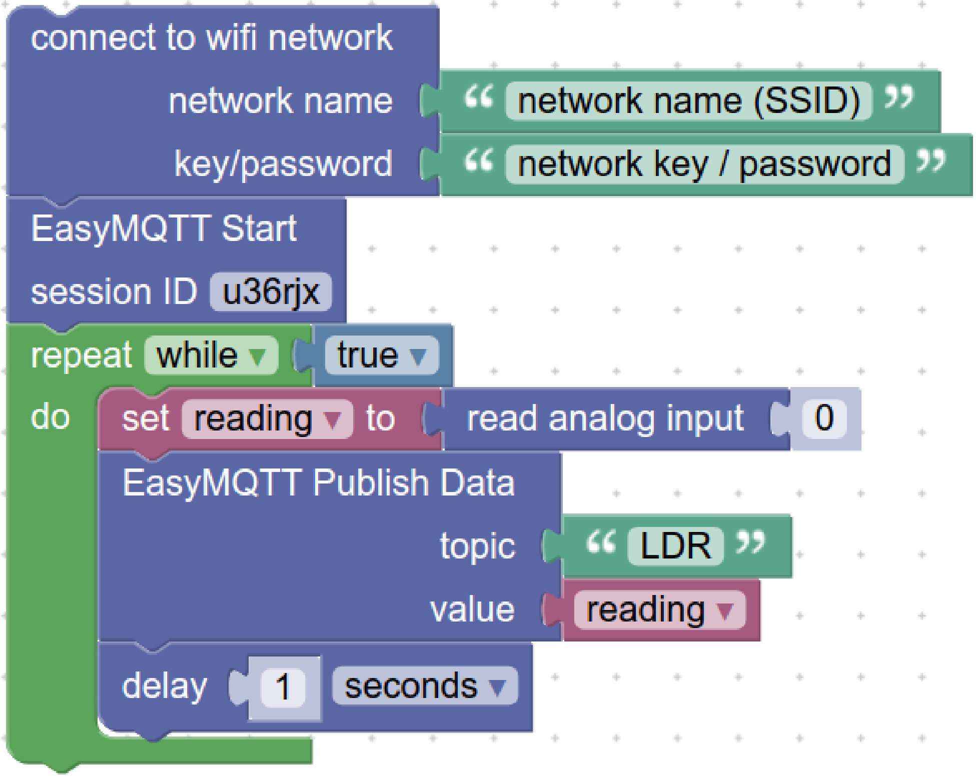
Networking: Sending data to the Internet/Cloud

The figure below illustrates a typical scenario of an application that sends data to a remote computer, having adequate software for receiving, storing, and making available these received data. This computer is called a **server** and is represented in the lower left part of the figure and, thanks to the Internet, it can be anywhere in the world. Such a server can be a physical or virtual machine, and it can also be active in “cloud” services such as Google Cloud Computing (GCC) or Amazon Web Services (AWS). The BIPES project server, for example, is hosted on Google's cloud platform. The figure also shows that various devices such as computers, tablets, and smartphones can access data stored on this server thanks to Internet connectivity. The figure also illustrates the embedded system, based on the ESP8266 card, which connects to the Internet through a wireless access point (Access Point).

In that way, an ESP8266 module can collect some sensor information, send the data to a server, and this server can store the received data and share it with other devices via a web page, for example.



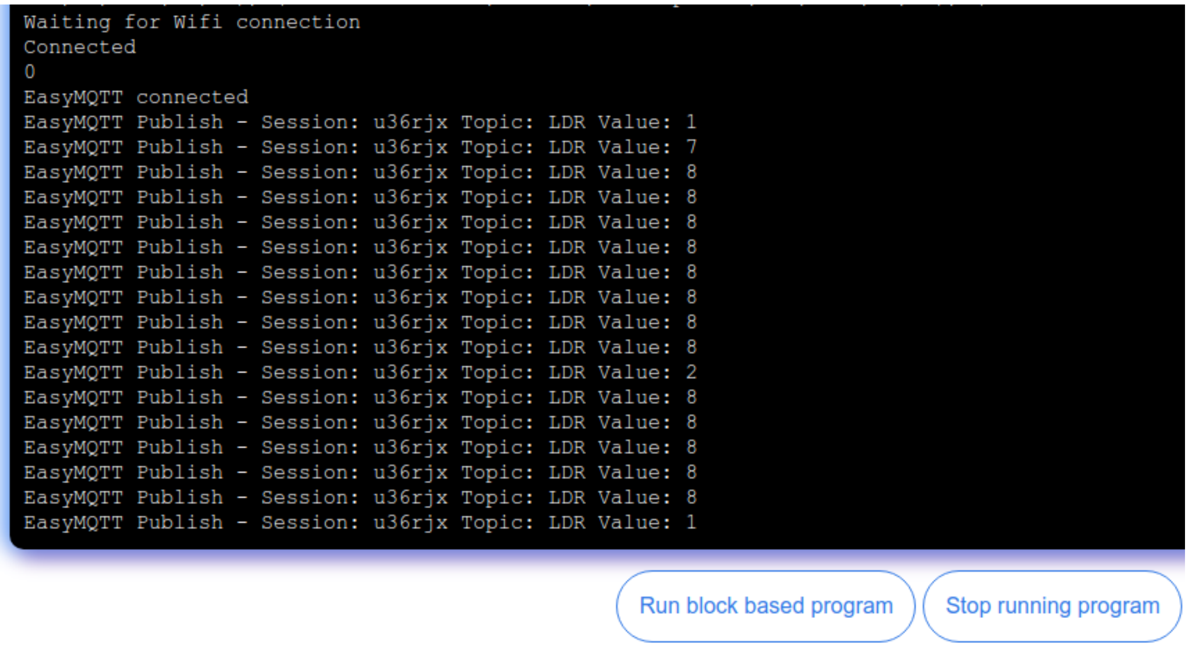
With the board connected to the Internet, use the blocks on the toolbox under **Network and Internet >> EasyMQTT** to create a program that sends data to the cloud. Define the "Topic" (*Topic*) and the data source or value (*Data*). The session ID (*Session ID*) is created automatically. However, you can manually define the Session ID if needed. For example, the following program sends analog readings from the LDR to the cloud using MQTT:



```
connect to wifi network
        network name  "network name (SSID)"
        key/password  "network key / password"
EasyMQTT Start
session ID u36rjx
repeat while ▾ true ▾
do  set reading ▾ to  read analog input 0
    EasyMQTT Publish Data
                   topic  "LDR"
                   value  reading ▾
    delay 1 seconds ▾
```

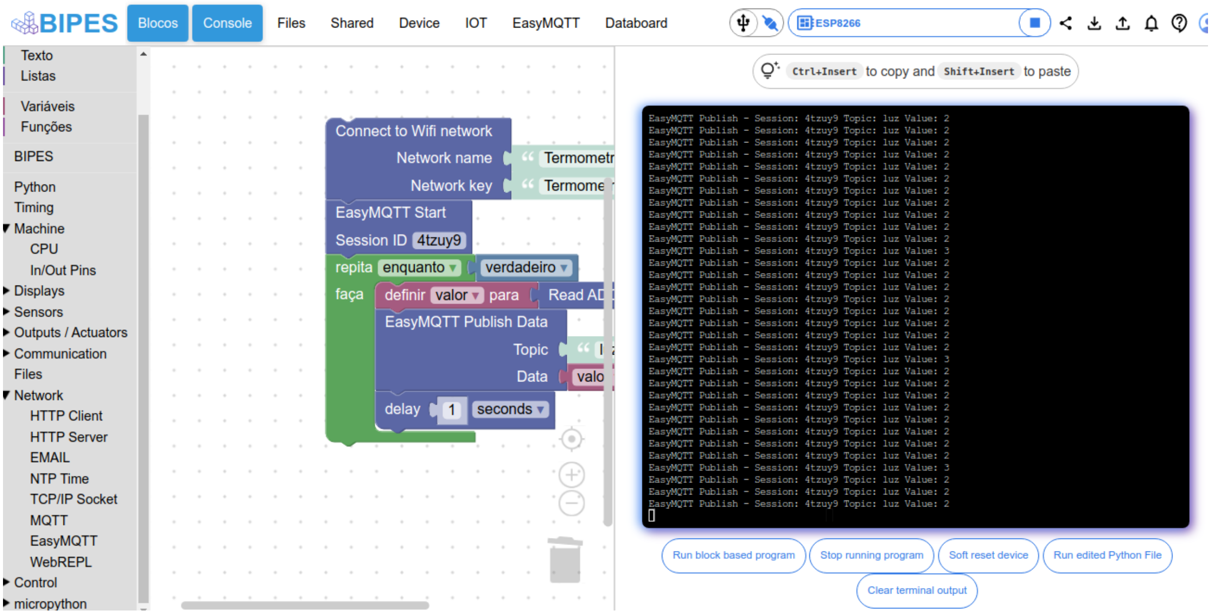
Message Queuing Telemetry Transport (MQTT) is a standard messaging protocol for the Internet of Things (IoT). In that way, BIPES offers the EasyMQTT feature, which provides an easy-to-use abstraction for MQTT services, allowing quick and flexible IoT prototyping. In addition, BIPES also enables you to create programs that communicate with any other MQTT device or service, such as ThingSpeak, ThingsBoard, or even SAP's IoT platform!

When running the program, you can see the result in the **Console** tab:



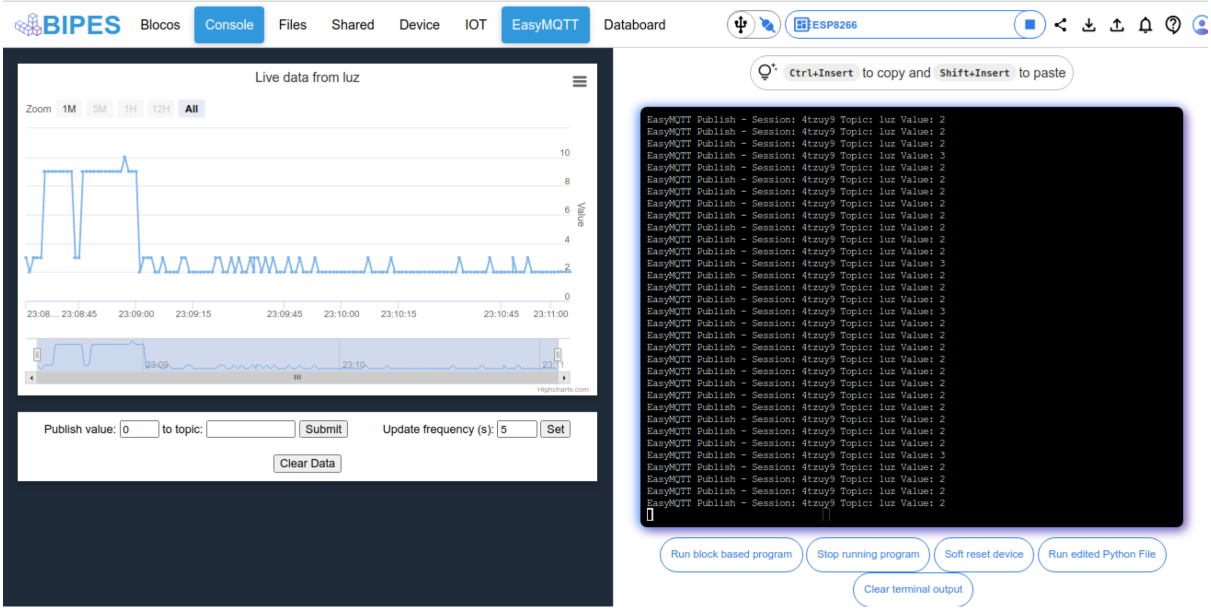
```
Waiting for Wifi connection
Connected
0
EasyMQTT connected
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 1
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 7
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 2
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 8
EasyMQTT Publish - Session: u36rjx Topic: LDR Value: 1
```

Remember that BIPES has a handy feature: **work/see two tabs simultaneously** by right-clicking with the mouse on one tab and left-clicking with the mouse on the other desired tab. Like this:



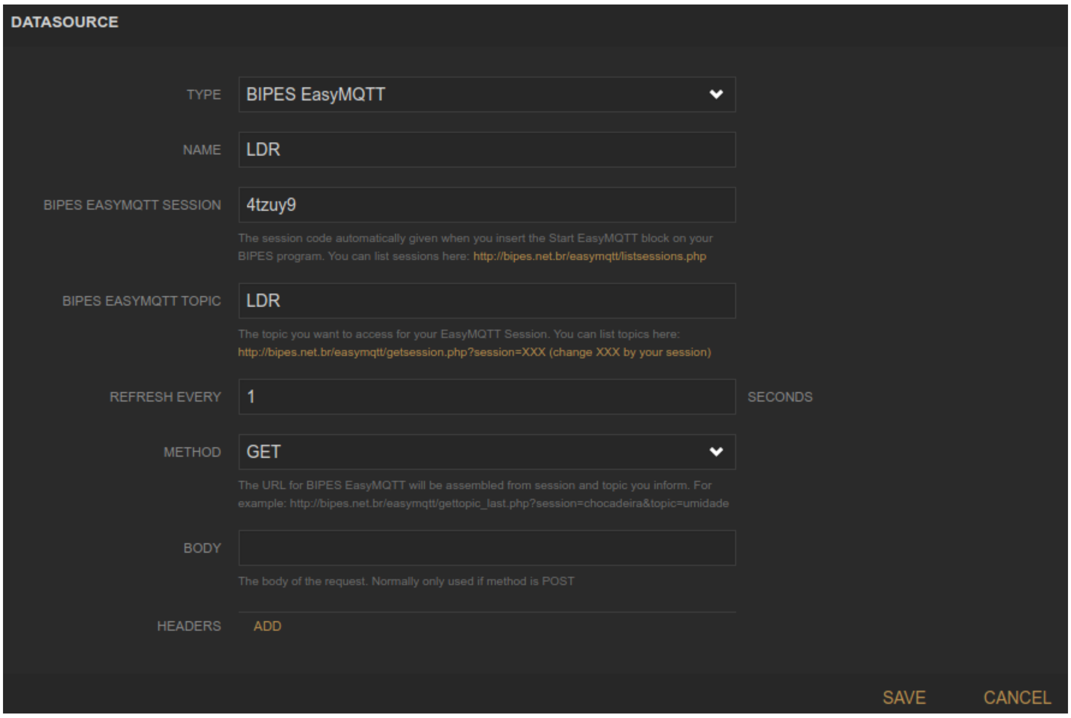
```
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 3
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 3
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 3
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
```

Thus, you can click the left mouse button on the **console** and the right mouse button on **EasyMQTT** to see the results sent to the cloud while you monitor the program execution.



```
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 3
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 3
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 3
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 3
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
EasyMQTT Publish - Session: 4tzuy9 Topic: luz Value: 2
```

Sharing: It is possible to click on the "button share" (<) and send an Internet link will be generated to share your program publicly. You can send this link to anyone, and when they access this link, they will be able to see real-time data sent by your board by clicking on the **EasyMQTT** tab. Finally, it's also possible to create a custom (*dashboard)* to visualize this data, using the IoT tab again (we've already done this for the USB cable, but now we'll do it using the Internet). This time, let's use the ***datasource*** EasyMQTT:



TYPE: BIPES EasyMQTT

NAME: LDR

BIPES EASYMQTT SESSION: 4tzuy9

The session code automatically given when you insert the Start EasyMQTT block on your BIPES program. You can list sessions here: http://bipes.net.br/easymqtt/listsessions.php

BIPES EASYMQTT TOPIC: LDR

The topic you want to access for your EasyMQTT Session. You can list topics here: http://bipes.net.br/easymqtt/getsession.php?session=XXX (change XXX by your session)

REFRESH EVERY: 1 SECONDS

METHOD: GET

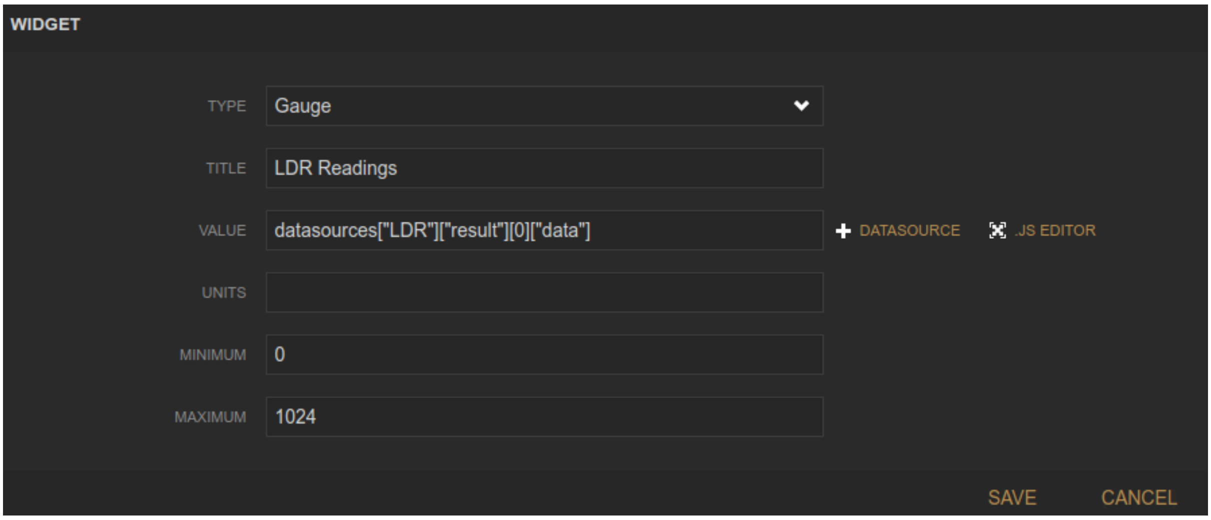
The URL for BIPES EasyMQTT will be assembled from session and topic you inform. For example: http://bipes.net.br/easymqtt/gettopic_last.php?session=chocadeira&topic=umidade

BODY:

The body of the request. Normally only used if method is POST

HEADERS: ADD

On the datasource, inform the EasyMQTT session and topic defined on your program and click SAVE. Next, let's add the panes and graphical widgets. First, add panes and then add widgets to your panes. For each widget, enter the datasource value. You can click on the "**+** DATASOURCE" button by the side of the value field for autocomplete to help you:

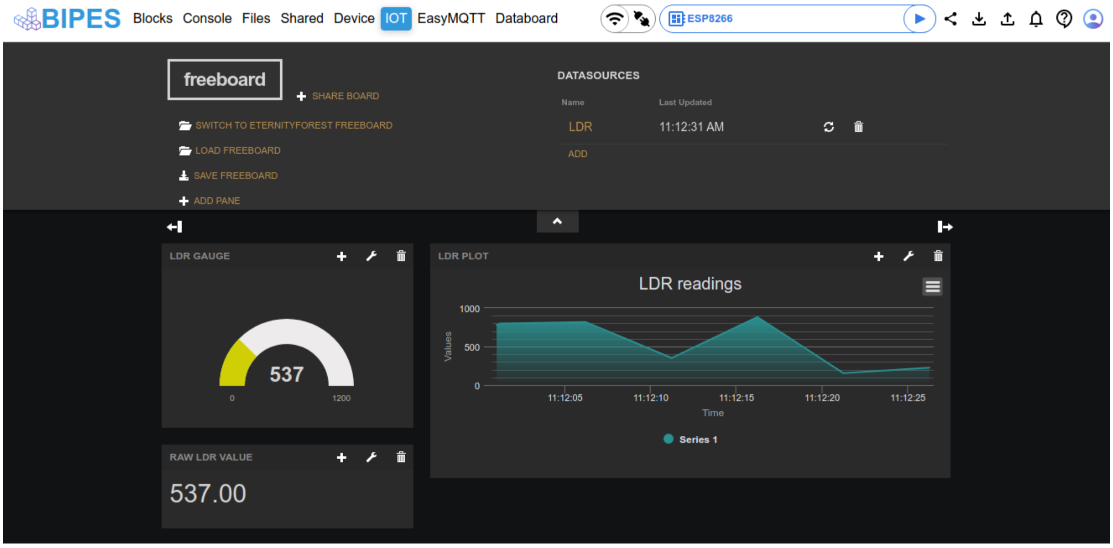


WIDGET

TYPE: Gauge
TITLE: LDR Readings
VALUE: datasources["LDR"]["result"][0]["data"]
UNITS:
MINIMUM: 0
MAXIMUM: 1024

In the example, the datasource used is: **datasources["LDR"]["result"][0]["data"]**

You can add other components and customize your dashboard. For example:



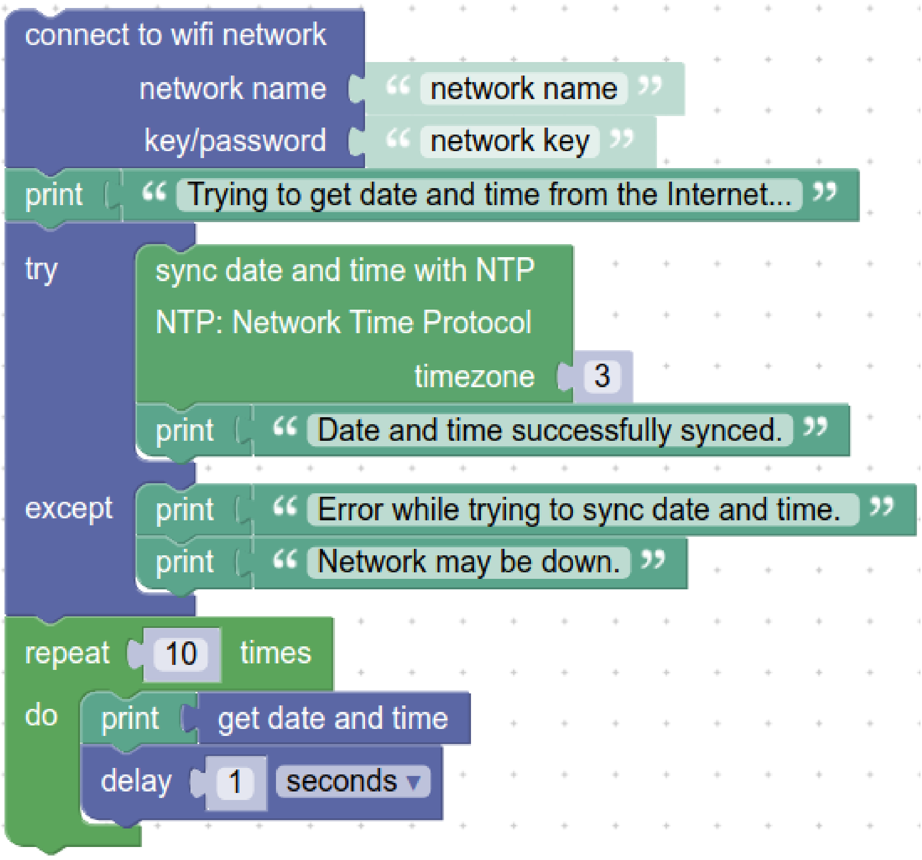
Again, note that you can share this program with another person via the link, using the button < . That person will have access to the dashboard and data sent by your device from anywhere in the world. The share option will share the program and access to the IoT and EasyMQTT tabs.

Handling errors

Several unexpected situations can occur in an embedded system, and in many cases, there will not be a user to reset the system. In addition, some embedded systems may be in inaccessible locations.

Thus, it is important to have ways to detect and handle errors. The block **try-except**, available in the **Python** toolbox, allows the system to try to execute an operation, and if the process fails, execute an alternative code. Such verification is helpful for several systems. For network-connected applications, it is also helpful because, for various reasons, the connection may be lost unexpectedly. Further, in some cases, network connection failure can interrupt the execution of the whole program until a user/operator restarts.

The following program illustrates an example of date and time synchronization from the Internet using the NTP block (available on the **Network and Internet** toolbox**)**. Even if there is a communication failure with the NTP server, the program continues.

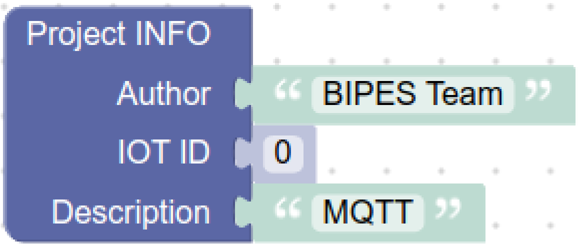


```
connect to wifi network
            network name  " network name "
            key/password  " network key "
print " Trying to get date and time from the Internet... "
try       sync date and time with NTP
          NTP: Network Time Protocol
                        timezone  3
          print " Date and time successfully synced. "
except    print " Error while trying to sync date and time. "
          print " Network may be down. "
repeat 10 times
do    print  get date and time
      delay  1  seconds
```

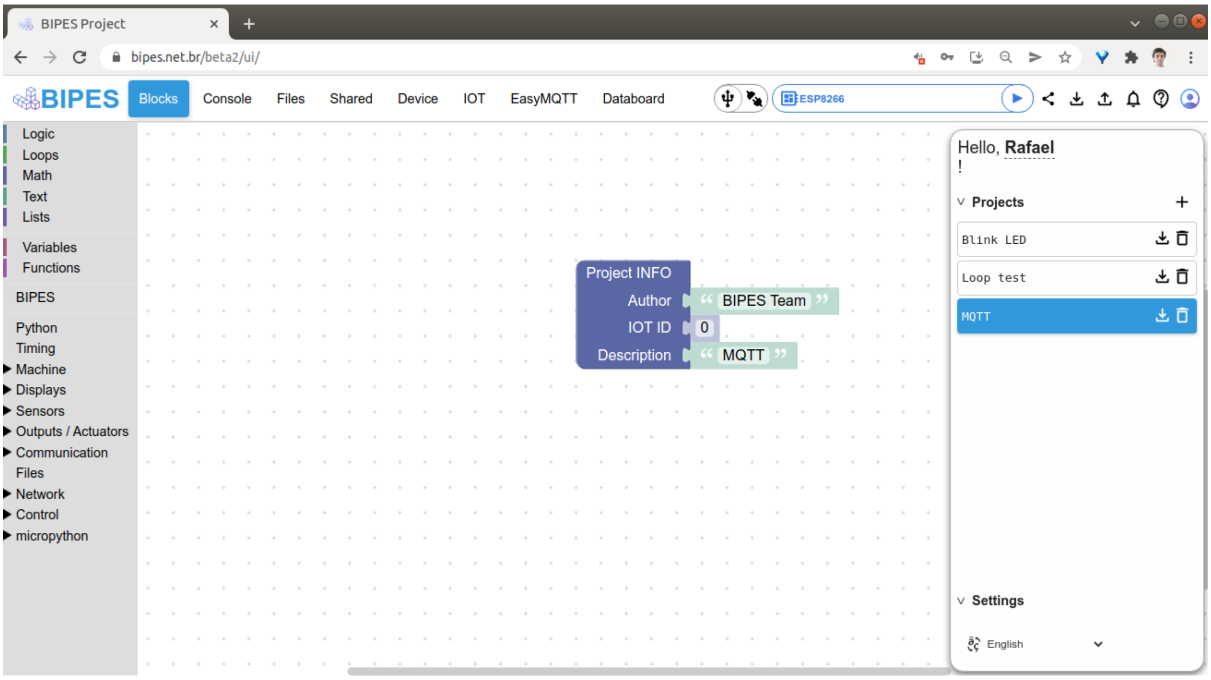
Without the structure **try**, if the program fails to connect to the NTP server, the program will never execute the blocks inside the structure **repeat**. Using the **try**, even if an error occurs, the program flow continues, and the repeat is executed.

BIPES: Multiple projects

With the ease offered by **BIPES**, you will not resist trying several programs and experiments. To facilitate this process, note that whenever you create a new program in **BIPES**, the following block is automatically made available on the desktop:



This block makes it possible to define the program’s name and author. Also note, in the upper right corner of the **BIPES** screen, that a profile icon can be used: 👤 . This icon will access the following **BIPES** area on the right area of the screen:

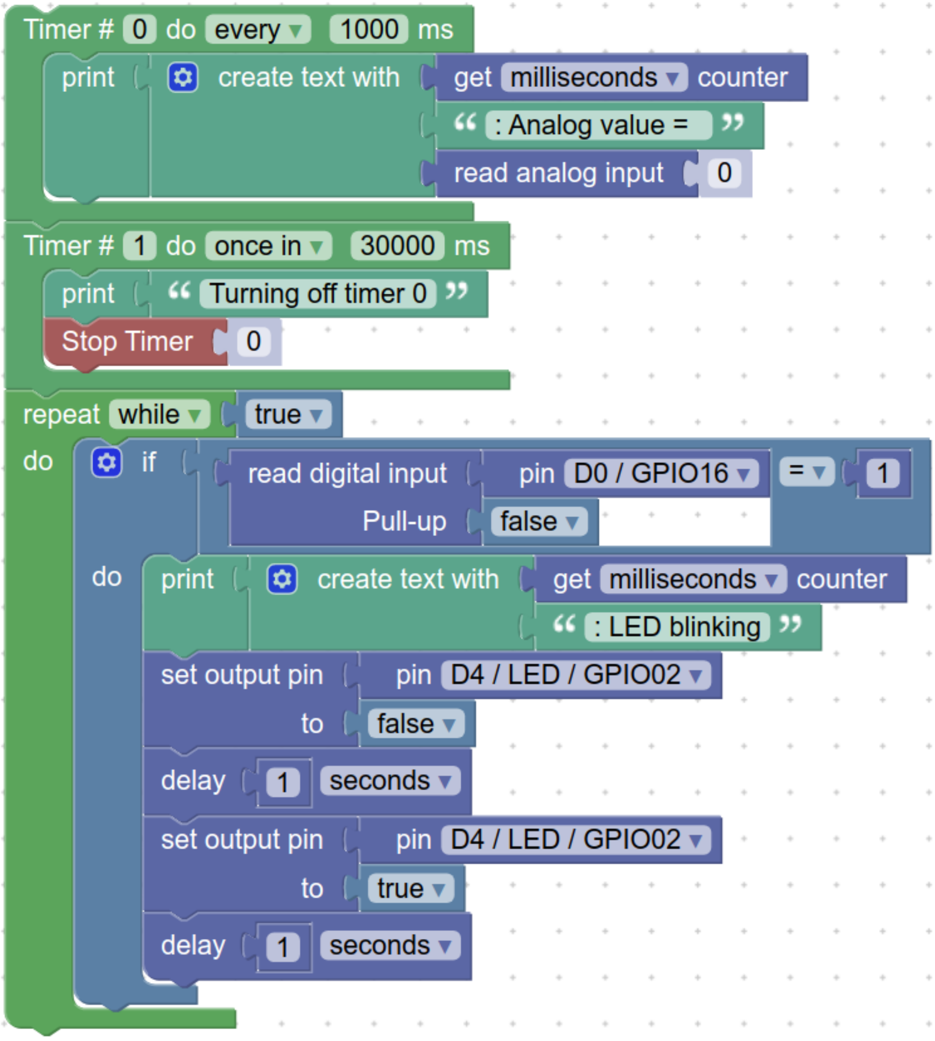


In this area, it is possible to create new programs, browse among already made programs, switch between programs, delete or download the XML file corresponding to that program to the local computer. The same area also allows you to change the language of the BIPES interface. Note that these programs are stored locally in your web browser storage area.

Parallel activities: *timers*

Another feature often used in embedded systems is timers. Timers are hardware-supported timing mechanisms that automatically generate events in scheduled time instants. Hardware timers generate interrupts in running programs, deviating the program's execution flow to a routine associated with a timer. When the timer routine is completed, the system returns the execution flow to the exact point where it diverted from the main program. As these switches occur quickly, the perceived effect is that the module is performing multiple tasks simultaneously.

The program below shows an example where ***Timer 0*** is configured to print the analog input reading on the **Console** every 1000 ms. Each line printed shows the count of time, in milliseconds, since the ESP8266 was turned on with the block **[get milliseconds counter],** followed by the analog reading. ***Timer 1*** is set to turn off the ***Timer 0*** after 30 seconds. The program also includes a repeat loop that flashes the LED on pin D4 while digital pin D0 has a high logic signal (1), showing the instant of time of occurrence of this event.



```
Timer # 0 do every 1000 ms
    print  create text with  get milliseconds counter
                             " : Analog value = "
                             read analog input 0
Timer # 1 do once in 30000 ms
    print " Turning off timer 0 "
    Stop Timer 0
repeat while true
do  if  read digital input  pin D0 / GPIO16  = 1
                            Pull-up false
    do  print  create text with  get milliseconds counter
                                 " : LED blinking "
        set output pin  pin D4 / LED / GPIO02
                        to false
        delay 1 seconds
        set output pin  pin D4 / LED / GPIO02
                        to true
        delay 1 seconds
```

The listing below shows the **Console** output after this program has run for a few seconds. The texts between the symbols “<< >>” indicate our notes, which were not printed on the **Console** by the running program.

```
===
<< PROGRAM STARTED WITH PIN D0 IN LOW (0) LOGIC LEVEL (D0 ⇔ GND)>>
3816: Analog value = 1
4816: Analog value = 1
5816: Analog value = 1
6816: Analog value = 1
7816: Analog value = 1
8816: Analog value = 1
9816: Analog value = 1
10816: Analog value = 1
11816: Analog value = 1
12816: Analog value = 7
13816: Analog value = 7
14816: Analog value = 8
<< PIN D0 RECEIVES HIGH LOGIC LEVEL (1) (D0 ⇔ 3V3)>>
15445: LED blinking
15816: Analog value = 7
16816: Analog value = 7
17453: LED blinking
17816: Analog value = 8
18816: Analog value = 8
19461: LED blinking
19816: Analog value = 8
20816: Analog value = 8
<< SOME SECONDS OF DATA SHOWN ON THE CONSOLE WERE OMITTED >>
31510: LED blinking
31816: Analog value = 1
32816: Analog value = 8
Turning off timer 0
<< AFTER 30 SECONDS, TIMER 0 IS TURNED OFF AND THE LOOP CONTINUES >>
33518: LED blinking
35526: LED blinking
37534: LED blinking
39543: LED blinking
41551: LED blinking
43559: LED blinking
```

The numbers presented before the “:” show the time, in milliseconds, of each event. Note that the events of the main loop and the two timers (*timers*) occur independently**.**

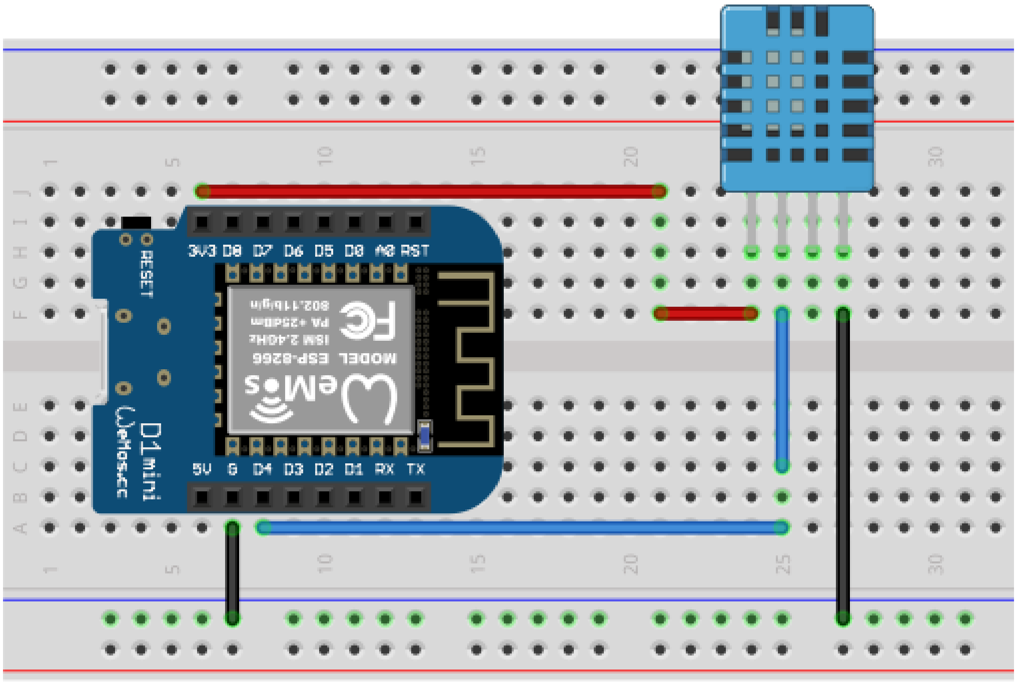
Temperature and humidity sensor

We can also include a temperature and humidity sensor in the system and send measurements to the cloud! Use *jumper cables directly*, or use the *protoboard* to connect the DHT11 humidity and temperature sensor to the ESP8266 module. Note that the sensor must be connected to pin 3V3 to receive power, to the ground (GND), and the output (second pin) to input pin D1 or D4 (or whatever you find suitable) on the board.

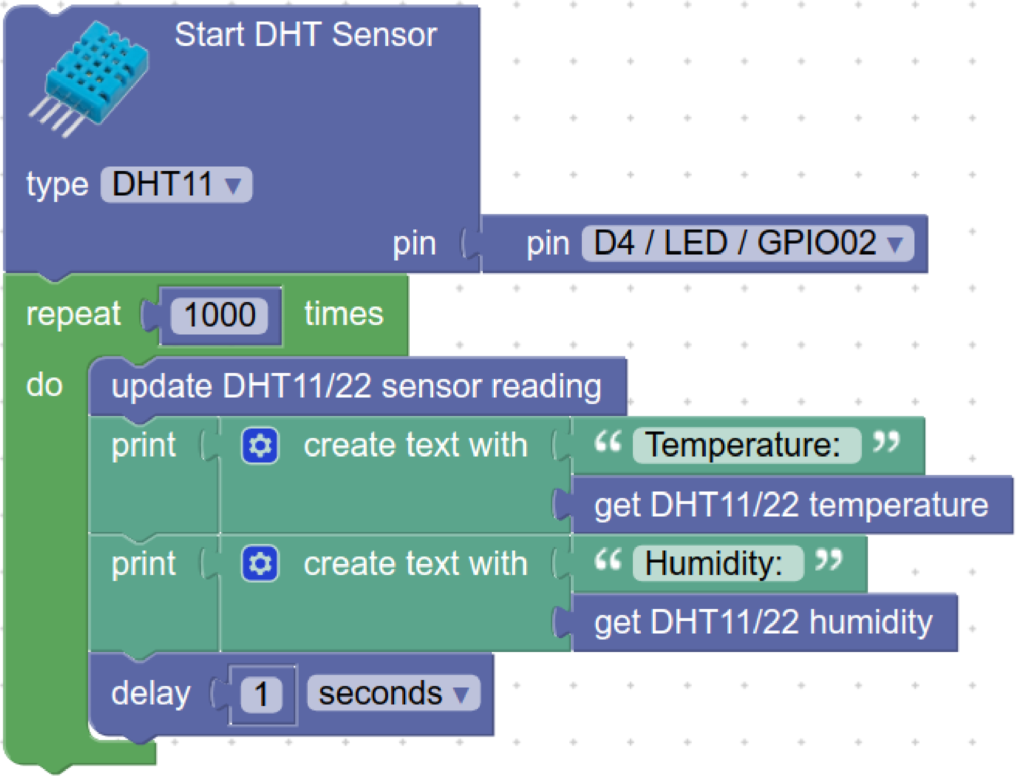
Connection suggestion:

Pin of DHT11	**Pin of ESP8266**
GND (Pin 4)	**G**
VCC (Pin 1)	**3V3**
DATA (Pin 2)	**2 (D4)**

Circuit assembly suggestion for the board WeMos D1 mini board with the module ESP8266 connected to DHT11:

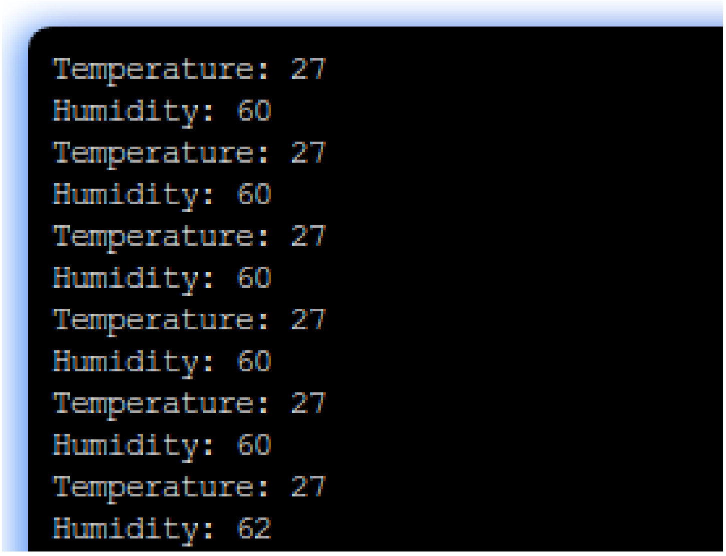


Program:



```
Start DHT Sensor
type DHT11
pin   pin D4 / LED / GPIO02
repeat 1000 times
do  update DHT11/22 sensor reading
    print  create text with  " Temperature: "
                             get DHT11/22 temperature
    print  create text with  " Humidity: "
                             get DHT11/22 humidity
    delay 1 seconds
```

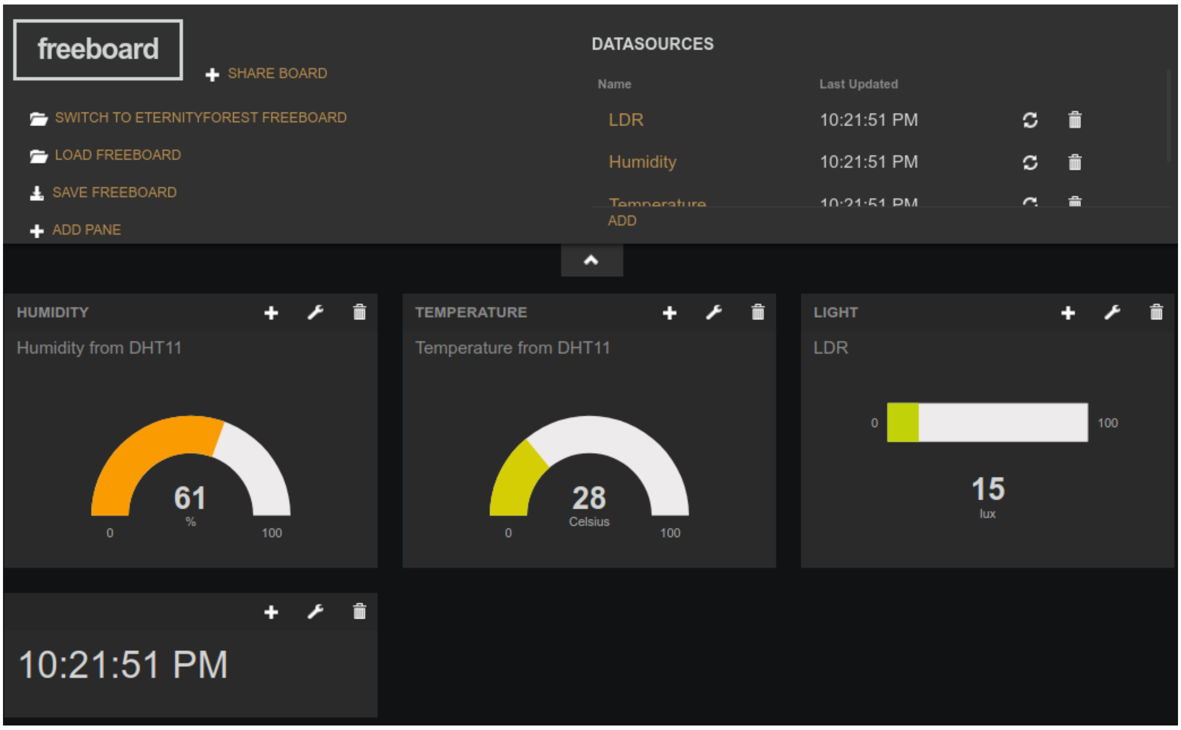
Run the program, and view the result in the **Console** tab. Example result:



```
Temperature: 27
Humidity: 60
Temperature: 27
Humidity: 60
Temperature: 27
Humidity: 60
Temperature: 27
Humidity: 60
Temperature: 27
Humidity: 60
Temperature: 27
Humidity: 62
```

Activity:
Use EasyMQTT to include one additional feature to send the humidity and temperature to the cloud, along with the readings from the LDR. In the IoT tab, configure an IoT (*dashboard*) and monitor readings sent by the board using another device/computer.

Example result from the activity:

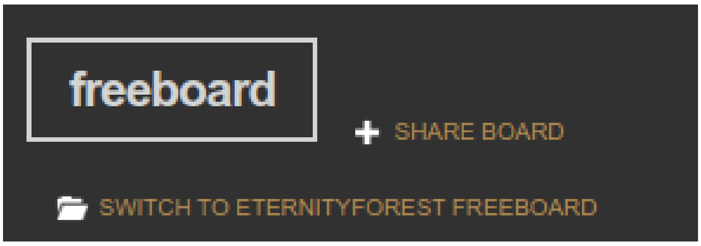


Name	Last Updated
LDR	10:21:51 PM
Humidity	10:21:51 PM

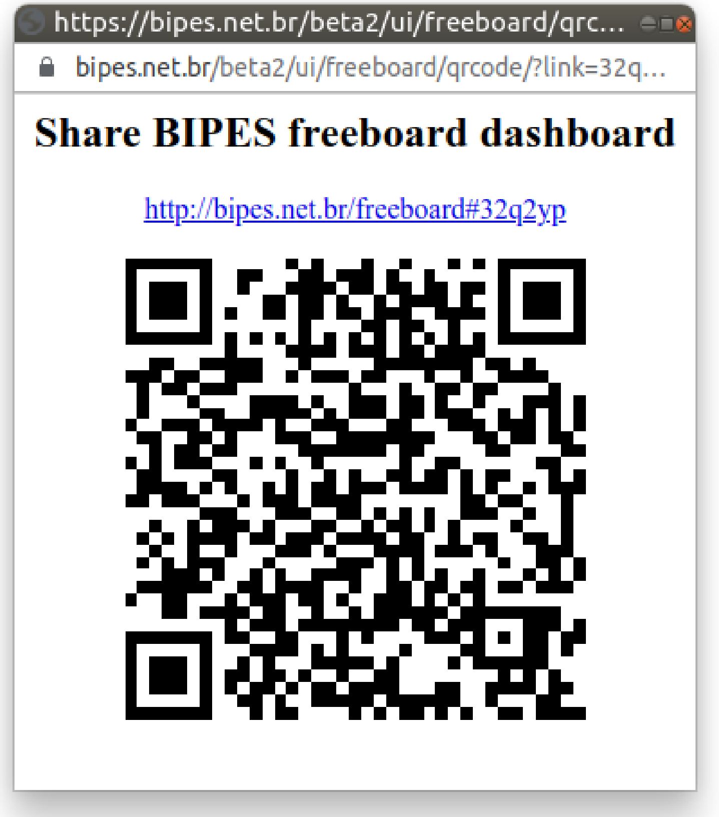
Share the dashboard with smartphones and other devices

After the application is ready, it is possible to share the link only for the visualization panel (**freeboard** *dashboard*) with any device, including optimization for mobile phones (responsive web technology).

Click on the share button ⋖ of BIPES, and a link will be generated, similar to: http://bipes.net.br/beta2/ui/#4c6bfm. Replace the term *beta2/ui/* with *freeboard* so that the new link will be: http://bipes.net.br/freeboard#4c6bfm. This last link will only share the *dashboard* (freeboard). Another possibility is to use the option **SHARE BOARD,** next to the **freeboard** icon:



When you click on the "**+ SHARE BOARD**" button, a link is generated with its associated QR-Code, and both will be displayed in a new window, as shown below. Note that you may have to authorize pop-up windows on your web browser.

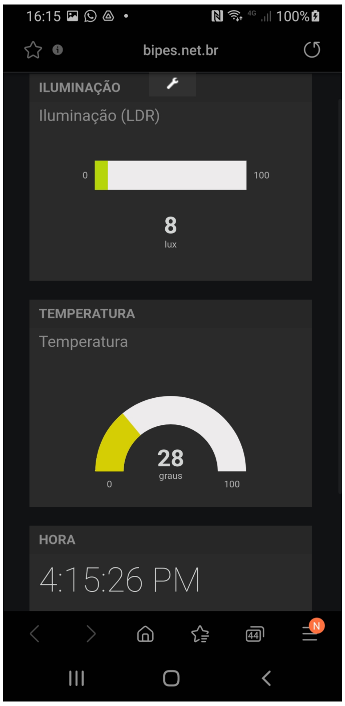

https://bipes.net.br/beta2/ui/freeboard/qrc...
bipes.net.br/beta2/ui/freeboard/qrcode/?link=32q...

Share BIPES freeboard dashboard

http://bipes.net.br/freeboard#32q2yp

Now you can share this link or QR-Code with any user who will view the data and even send commands to your system (if you include an interactive application with buttons (*switches*)). In addition, when accessing the QR-Code, or link, by cell phone, the user also has access to a responsive dashboard, similar to the one on the right, thanks to freeboard!

To learn more about freeboard, check the freeboard GitHub repository: https://github.com/Freeboard/freeboard

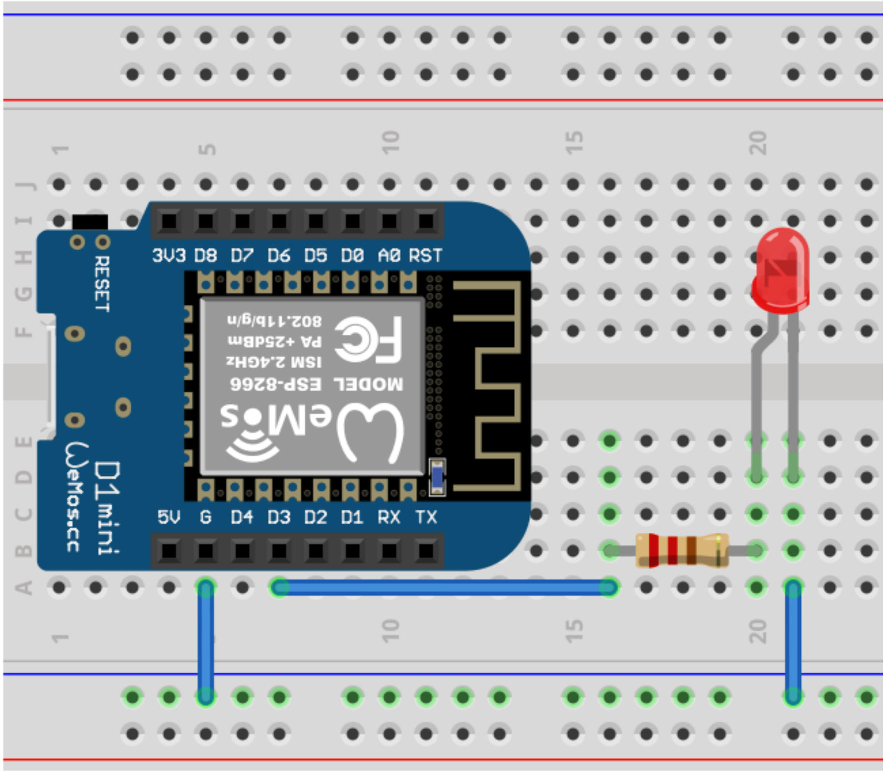


PWM: LED Brightness Control

We've already discussed how to control a digital output and check the status of digital input and analog input pins. Another possibility is to use the digital output of the PWM type (**Pulse Width Modulation**), which allows a digital signal in the form of a square wave to approximate a voltage value equivalent to an analog output. Thus, it is possible to use a pin with PWM output to control the brightness of an LED, the speed of a motor, the temperature of a heating resistor, among other possibilities. Unfortunately, not all pins support PWM output, so the LED already included on the ESP8266 board is not connected to a PWM pin. Therefore, we will need an external LED connected to a pin with a PWM output feature to be able to test this feature. In the example below, we use pin D3 (GPIO0).

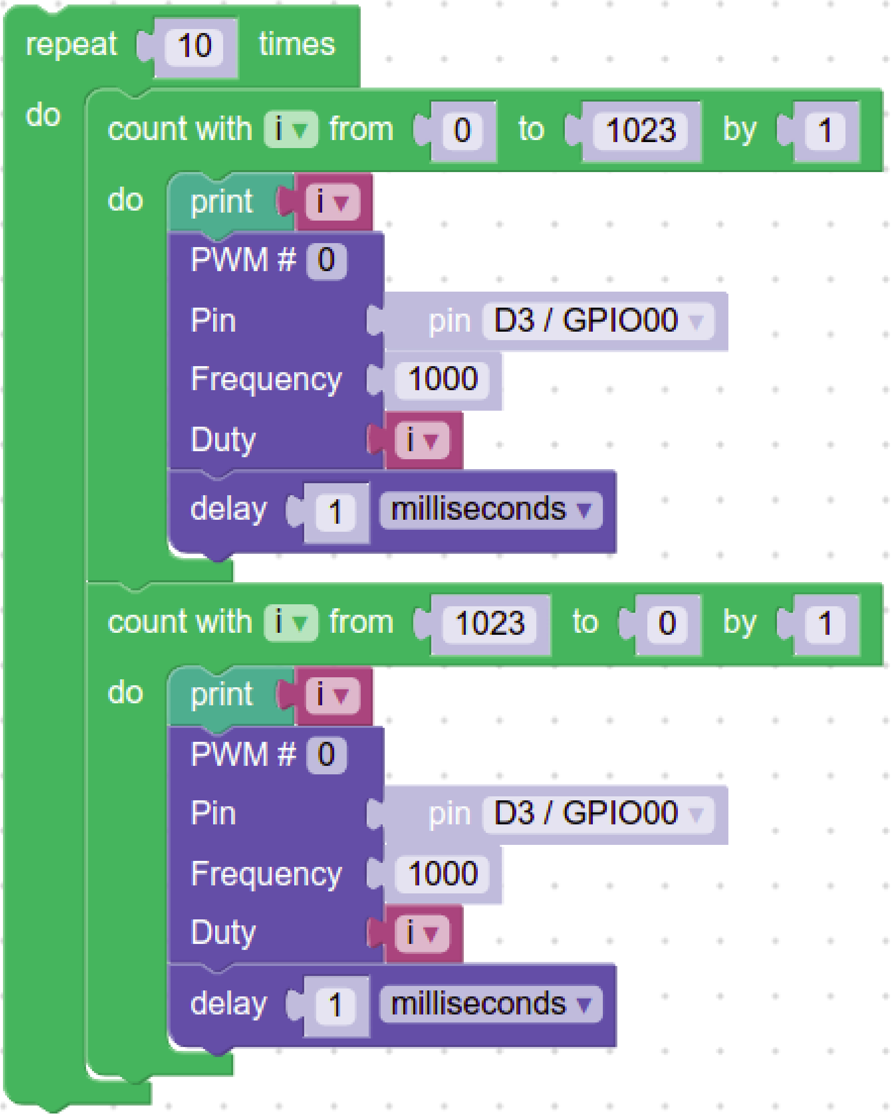
Circuit assembly:

(**Caution**: Note that the LED has a polarity, ie, positive (more extended terminal) and negative (shorter terminal, next to the bevel in the LED) pins. Therefore, the LED's negative terminal must be connected to G (ground) and the positive terminal to the resistor, which is connected to pin D3).



The PWM has two main parameters: the frequency of the generated square wave signal (**Frequency**) and the *duty cycle* **(Duty)**. Right now, we are interested in the **duty cycle**. This parameter indicates the portion of the square wave at a high level (on), which can vary from 0 to 100%. Thus, a 100% duty cycle means full power, 50% equals half power, and 0% no power. On ESP8266 the PWM has 10 bit resolution (going from 0 for 0% to 1023 for 100%).

The program below makes the LED gradually increase its brightness from 0 to maximum and decrease its brightness progressively to 0, repeating this process ten times. Note how, in this application, the value of the parameter **duty** varies with the count of the variable *i*.



```
repeat 10 times
do  count with i from 0 to 1023 by 1
    do  print i
        PWM # 0
        Pin        pin D3 / GPIO00
        Frequency  1000
        Duty       i
        delay 1 milliseconds
    count with i from 1023 to 0 by 1
    do  print i
        PWM # 0
        Pin        pin D3 / GPIO00
        Frequency  1000
        Duty       i
        delay 1 milliseconds
```

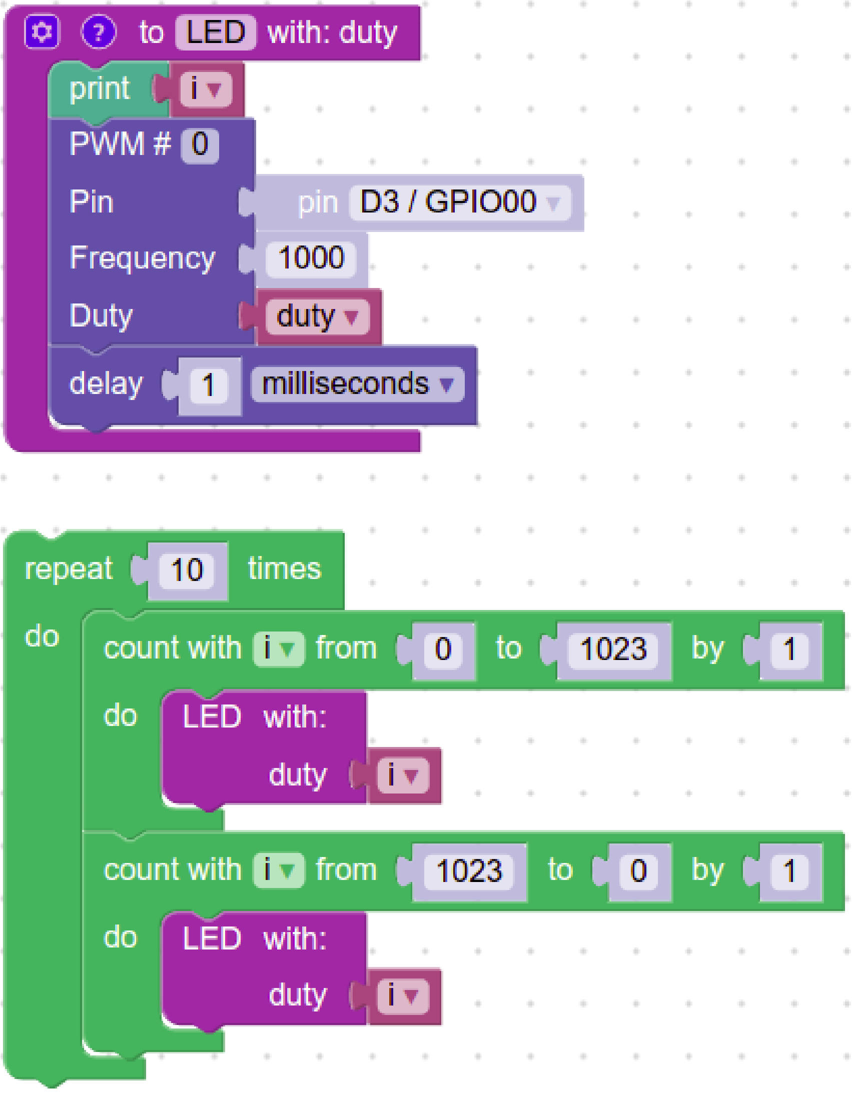
The program above makes the LED increase its brightness smoothly and slowly from 0 to the maximum brightness, and then it reduces its brightness smoothly until 0, repeating this process ten times.

Activity:
Use EasyMQTT and the IoT tab and implement a *slider* (slider) on the IoT tab that allows you to control the LED brightness from a mobile phone or another device. Then control the LED brightness from your mobile phone!

Tip: Use the block **subscribe to a topic** from EasyMQTT blocks. More details about this practice are given in the following section: “Controlling devices via the Internet”.

BIPES: Subroutines / functions

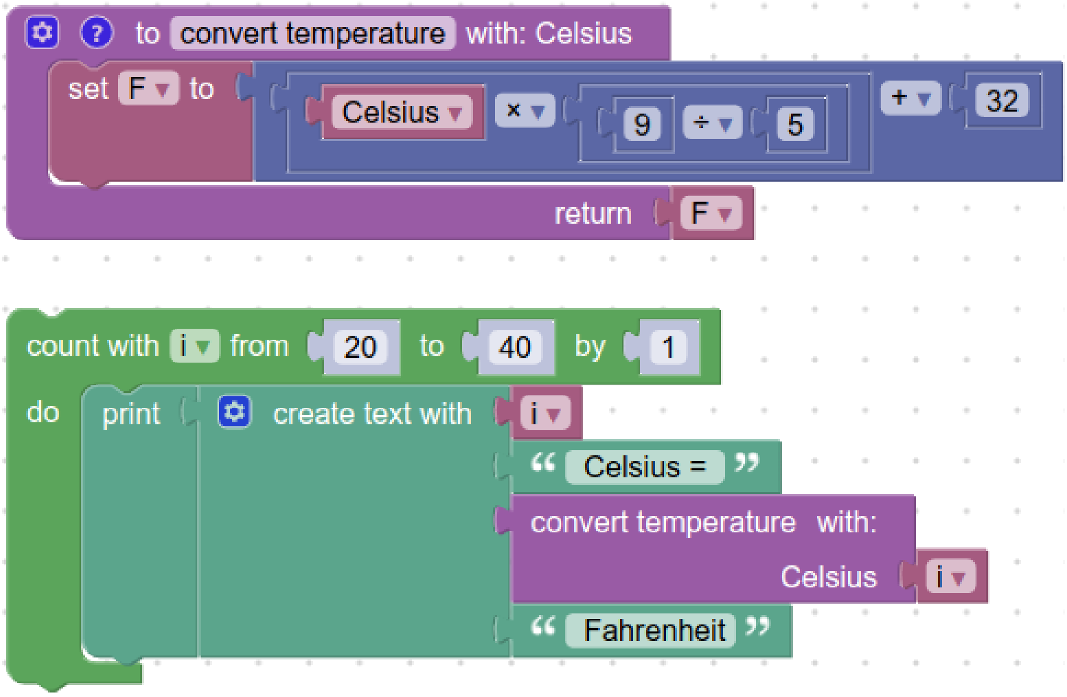
As your program gains new features, some code snippets may be helpful for reuse, making it worthwhile to create reusable blocks: these are the functions/subroutines that can be developed with the “**functions”** of the lateral toolbox. If we go back to the previous PWM and LED example, notice that three functions are repeated two times. These three functions could be grouped into a function and reused several times, as in the example below:



```
to LED with: duty
    print i
    PWM # 0
        Pin  pin D3 / GPIO00
        Frequency 1000
        Duty duty
    delay 1 milliseconds

repeat 10 times
do  count with i from 0 to 1023 by 1
    do  LED with:
            duty i
    count with i from 1023 to 0 by 1
    do  LED with:
            duty i
```

In the above example, we have created the **LED function** in the upper block area, which takes a duty parameter used internally in those blocks. Then, in the lower blocks, which constitute the main program, we call the **LED function** with the desired parameter (value of variable i). On each call, all blocks within the function are executed.

You can create various functions, of several complexities, with or without receiving parameters and create functions that return values. For example, a temperature unit conversion function could be prepared and used as follows:



And the result in the **Console** will be:

```
===
20 Celsius = 68.0 Fahrenheit
21 Celsius = 69.8 Fahrenheit
22 Celsius = 71.6 Fahrenheit
23 Celsius = 73.4 Fahrenheit
24 Celsius = 75.2 Fahrenheit
25 Celsius = 77.0 Fahrenheit
26 Fahrenheit = 78.8 Fahrenheit
27 Fahrenheit = 80.6 Fahrenheit
28 Fahrenheit = 82.4 Fahrenheit
29 Fahrenheit = 84.2 Fahrenheit
30 Fahrenheit = 86.0 Fahrenheit
31 Fahrenheit = 87.8 Fahrenheit
32 Fahrenheit = 89.6 Fahrenheit
33 Fahrenheit = 91.4 Fahrenheit
34 Fahrenheit = 93.2 Fahrenheit
35 Celsius = 95.0 Fahrenheit
36 Celsius = 96.8 Fahrenheit
37 Celsius = 98.6 Fahrenheit
38 Celsius = 100.4 Fahrenheit
39 Celsius = 102.2 Fahrenheit
40 Celsius = 104.0 Fahrenheit
>>>
```

After creating the functions, you can share your program with the button < and use the functions created in other programs!

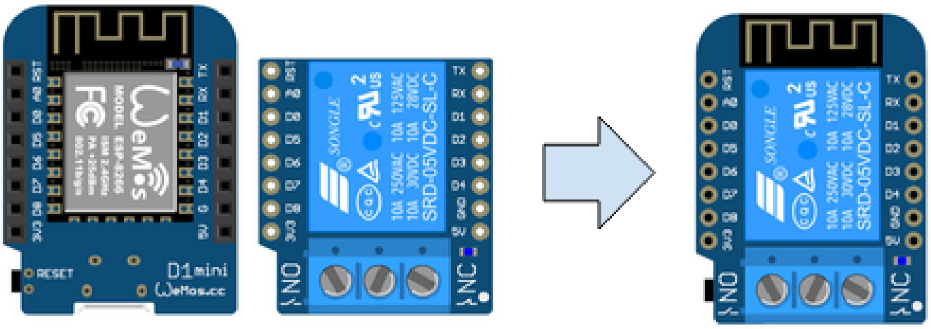
Controlling devices via the Internet

In this exercise, we will use a relay and electrical plugs to turn on / off any device connected to the socket (mains) via Web / Wifi. For testing purposes, we suggest controlling a light bulb. In that way, the device can be a lamp, a fan, a water pump, a heater, or any other device with its power rating supported by the relay. Of course, you will need a Wifi network/router for the connection. Suggestion: research on the functioning of the **relay** and the history of the **first computer "bug"**.

Caution:
This activity includes connecting the system to the mains power with voltages of 127 Volts or 220 Volts, presenting a risk of electric shock or other injuries. Only proceed if you feel safe or consult a specialized technician before connecting the system to the mains.

The relay is an electromechanical device that allows controlling electrical devices from digital signals. Thus, the ESP8266 board activates a relay module, triggering a load connected to an electrical outlet. For this activity, it is necessary to connect the relay to the ESP8266 board and the external electrical load to the relay.

One possibility is to use a relay module of the type "*shield*", which fits directly onto the ESP8266 board (WeMos D1 mini model), not requiring connection cables, as in the example below, where the two boards showed can be stacked, fitting them through their connection terminals.



There are three screw terminals on the relay board, where the load to be controlled can be connected. We note here that the relay operates as an electronically controlled switch, as it has three load control terminals: C (Common), Normally Open (NO), and Normally Closed (NC). You must use 2 of these terminals together (C with NO) or (C with NF). The choice of combination will define whether the relay turns the load on or off when activated by the ESP8266.

It is also possible to use a single or double relay module (allowing control of two or more devices) connected to the ESP board by jumper wires.

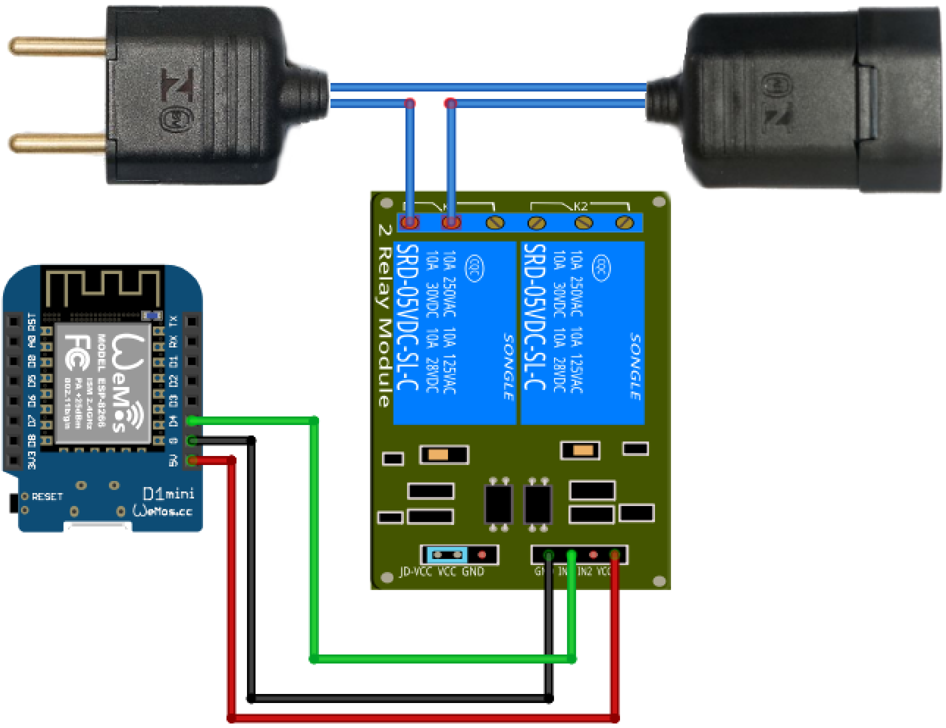
Relay connections with the ESP8266 module:

Relay module	Board with ESP8266 module
S or IN1 of the relay board	Pin D4 of the ESP8266 (control signal)
+ of the relay board	Pin VIN (5 Volts) of the ESP8266 (power)
- of the relay board	Pin GND (earth) of the ESP8266

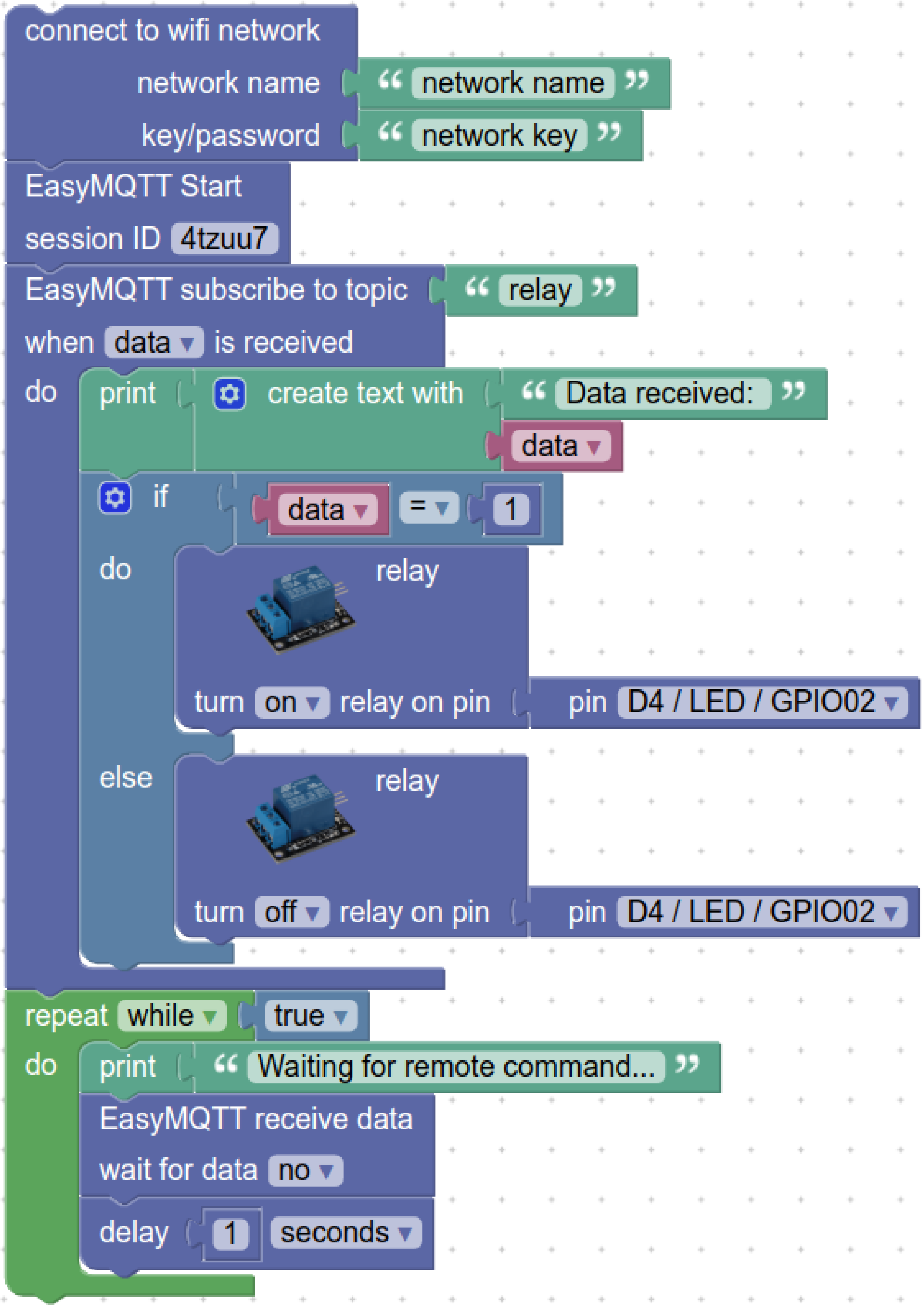
Attention: The entire assembly procedure must be carried out with the devices disconnected from the mains.

The diagram below illustrates a two-wire plug and socket assembly, similar to a power cord extension. One wire directly connects the plug and the socket, while the other wire passes through the relay (normally open: pins C and NO; or normally closed: pins C and NC).

Check the connections carefully, and if necessary, ask an electronics or electrical technician for assistance. Then, after careful verification and validation of the assembly, connect the load to the plug (we will use a lamp in this application) and, finally, connect the power plug to the mains.



Remember that the purpose of this practice is to activate the light bulb via the Internet. Therefore, the program in blocks for this application is as follows:



```
connect to wifi network
        network name  " network name "
        key/password  " network key "
EasyMQTT Start
session ID 4tzuu7
EasyMQTT subscribe to topic  " relay "
when data ▾ is received
do  print  create text with  " Data received: "
                             data ▾
    if  data ▾ = ▾ 1
    do    relay
          turn on ▾ relay on pin  pin D4 / LED / GPIO02 ▾
    else  relay
          turn off ▾ relay on pin  pin D4 / LED / GPIO02 ▾
repeat while ▾ true ▾
do  print  " Waiting for remote command... "
    EasyMQTT receive data
    wait for data no ▾
    delay 1 seconds ▾
```

To test the program, run it and go to the EasyMQTT tab. Type "**relay**", fill in the data with 0 or 1 and send the data. Then, check if the relay or LED changes its state between on / off at your commands.

Note that the remote control can be done from any device connected to the Internet that knows the session through a web address. BIPES also offers an Application Programming Interface (API) with web requests (HTTP) to integrate other applications. We will detail this topic in the next section.

For now, it is worth mentioning that BIPES HTTP API for integration allows other systems to change EasyMQTT topics values using HTTP requests. The update is done via the page **publish.php**, where the desired session, topic, and value must be provided. For example, for session **4tzuu7** and topic **relay**, the following requests can be used:

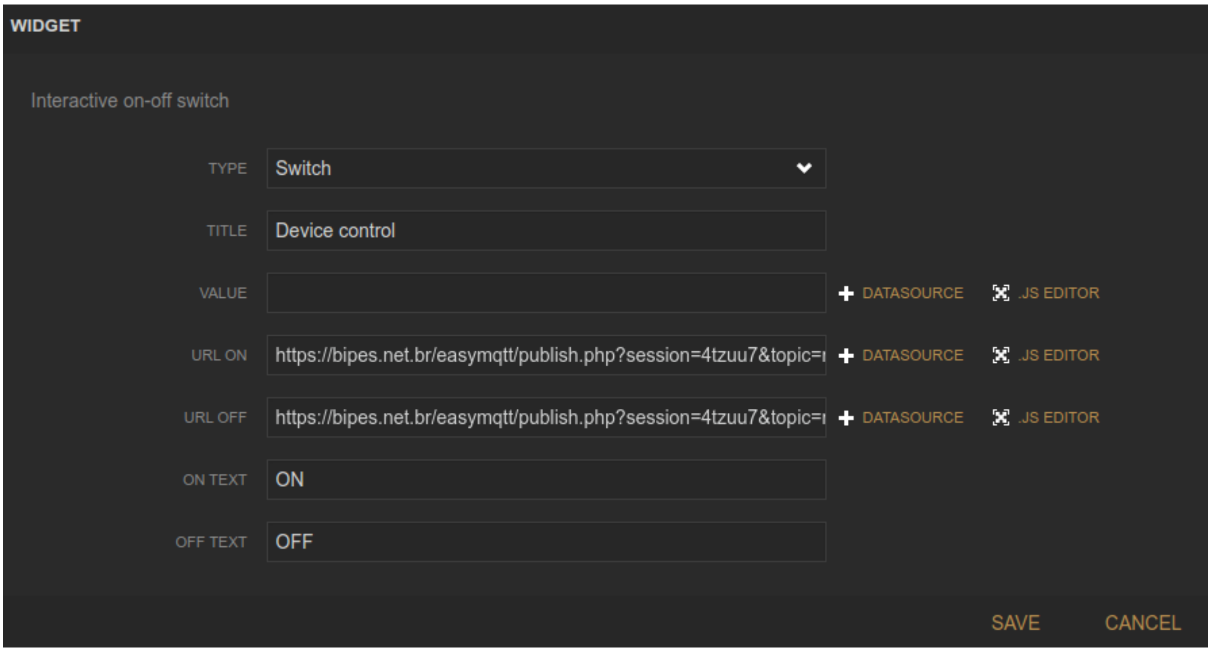
To turn on the device:

http://bipes.net.br/easymqtt/publish.php?session=4tzuu7&topic=relay&value=1

To turn off the device:

http://bipes.net.br/easymqtt/publish.php?session=4tzuu7&topic=relay&value=0

Note that each program will have a different session (**4tzuu7in** this case). Adjust the session, or create your own! You can include these web addresses in button actions on the IoT tab. Also, check if your BIPES access was via HTTP or HTTPS (check the browser URL). Those on/off addresses mentioned above must start with http or https, depending on how BIPES was accessed. For example, the configuration of a **Switch** widget on the freeboard dashboard (IOT tab) would look like this:



WIDGET

Interactive on-off switch

TYPE: Switch

TITLE: Device control

VALUE:

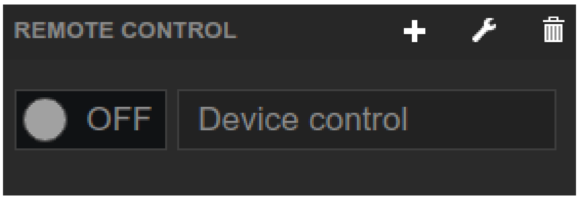
URL ON: https://bipes.net.br/easymqtt/publish.php?session=4tzuu7&topic=

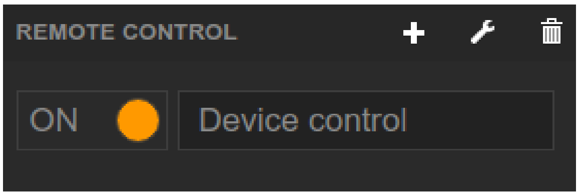
URL OFF: https://bipes.net.br/easymqtt/publish.php?session=4tzuu7&topic=

ON TEXT: ON

OFF TEXT: OFF

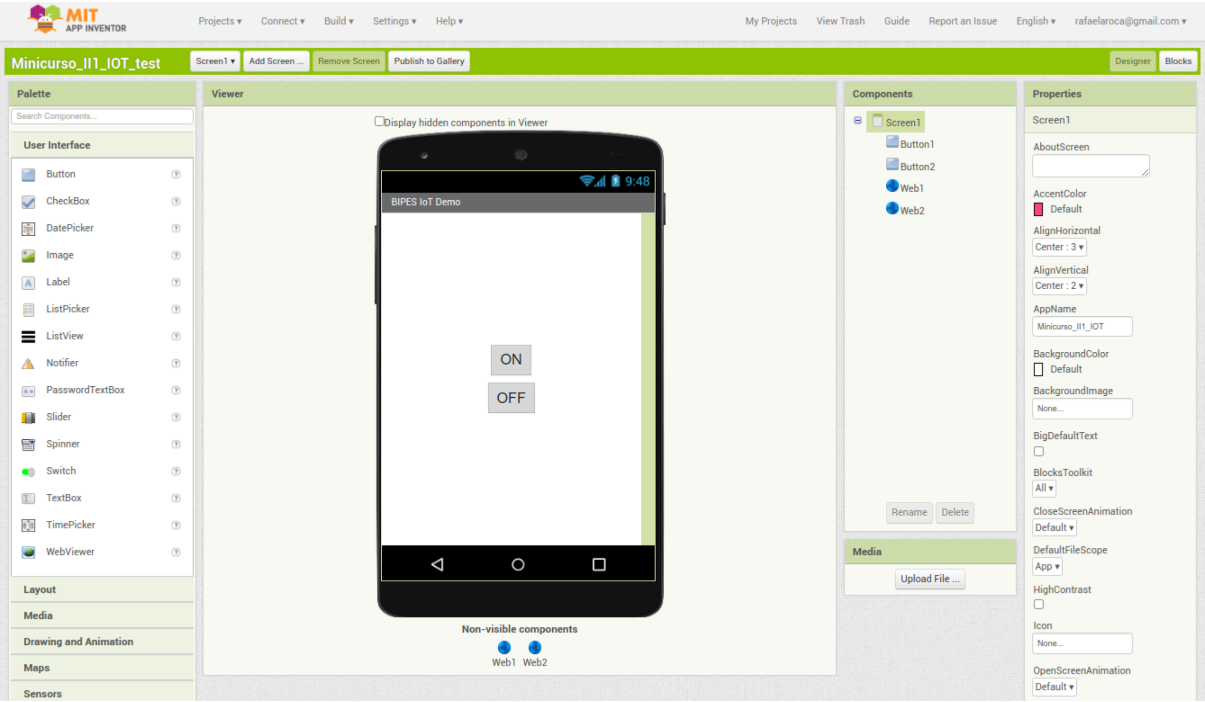
Now, by clicking the buttons on the IoT tab, you will control the device connected to the relay, such as a light bulb, for example:





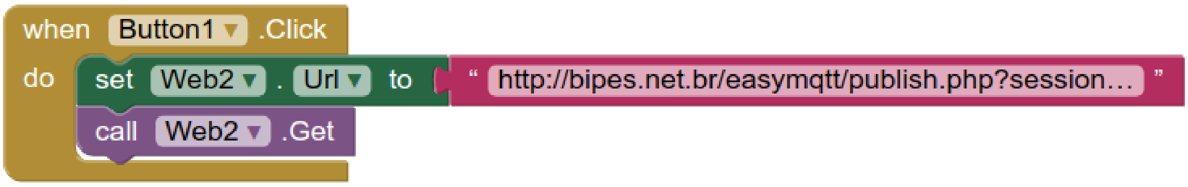
If you share this program with another person via the link, using the button , that person will have access to the control panel from anywhere in the world and control the device remotely.

You can also use MIT App Inventor (http://ai2.appinventor.mit.edu/) to create a smartphone app that controls the device. The complete application can be made *online* using the website http://ai2.appinventor.mit.edu/:

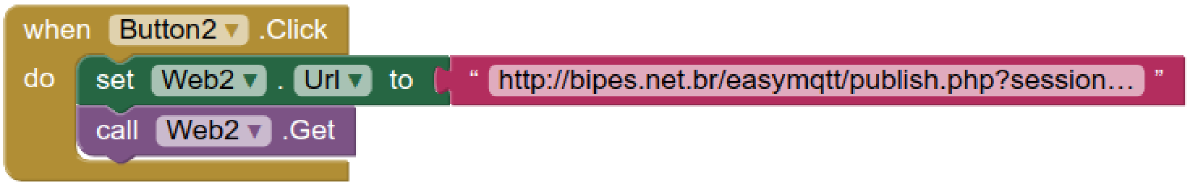


After creating an account and logging into MIT App Inventor, in the designer screen, create a new program and include two buttons: ON and OFF, as shown in the figure. If you want, you can add images, adjust sizes, colors, and texts to make your program user interface more attractive.

In blocks (click on the top right button to access “blocks”), include two blocks that define the actions of the buttons, each one configured to make HTTP requests to the links as mentioned earlier:



```
when Button1 .Click
do  set Web2 . Url to " http://bipes.net.br/easymqtt/publish.php?session... "
    call Web2 .Get
```



```
when Button2 .Click
do  set Web2 . Url to " http://bipes.net.br/easymqtt/publish.php?session... "
    call Web2 .Get
```

Done! The blocks above build a complete app for Android phones, which allows you to turn on/off devices remotely. Note how similar it is to BIPES and how easy to use! In fact, both BIPES and MIT App Inventor are based on the Google Blockly Programming language. That is why the blocks are so similar.

Activity:

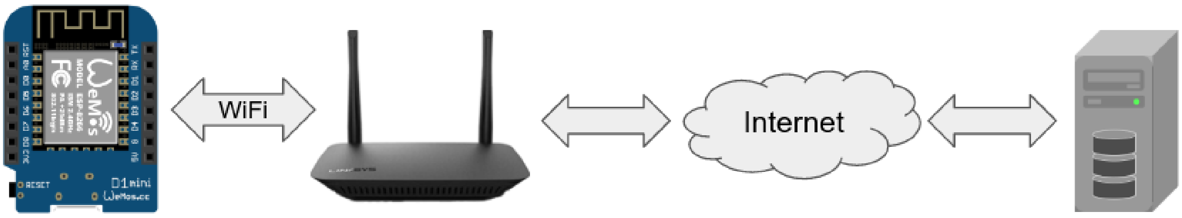
Remember the example mentioned in the introduction of this book: the "on/off" control of a refrigerator. You already have the knowledge and tools to implement this type of control!

Use the ESP8266 or ESP32 board and the assembly with the relay, made in the previous exercise, together with a DHT11 temperature and humidity sensor. Then, prepare a program that turns on a relay, responsible for activating a fan when the temperature exceeds 30 degrees Celsius, and turns off the relay when the temperature is below 25 degrees Celsius.

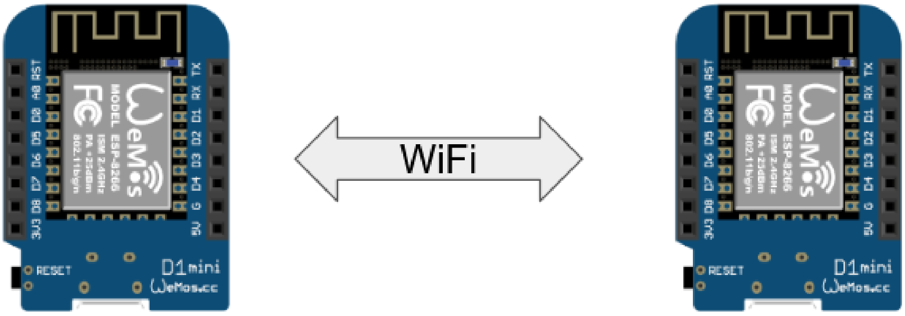
After testing the system, include the functionality to monitor the temperature remotely. The option to adjust, via the dashboard, the minimum and maximum temperature that keeps the relay, which controls the fan, activated.

Web client and web server (HTTP)

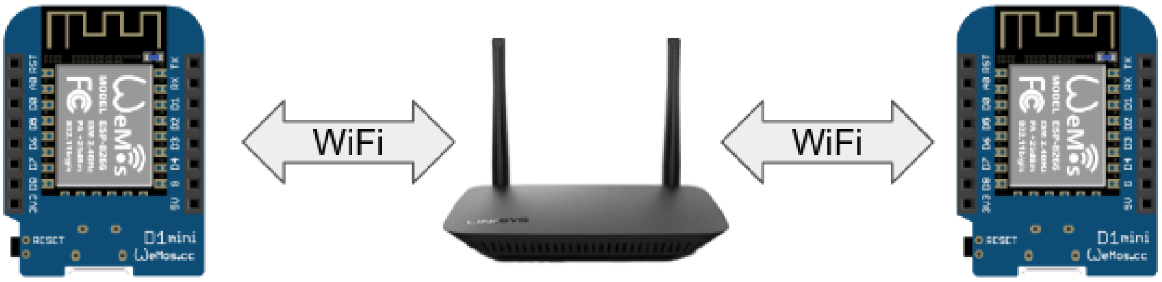
Nowadays, a widespread activity is accessing websites such as http:///www.bipes.net.br/. For this type of access, a web browser such as Google Chrome, for example, uses the hypertext transfer protocol - *HyperText Transfer Protocol* **(HTTP)**. The exchange of information over HTTP is initiated by a **client** device that starts the process through a request sent to the **server.** The following figure illustrates a possible situation where an ESP8266 board acts as a client, making a request to a server via the Internet. Note that such a scenario could also be reversed, where ESP8266 could act as a server.



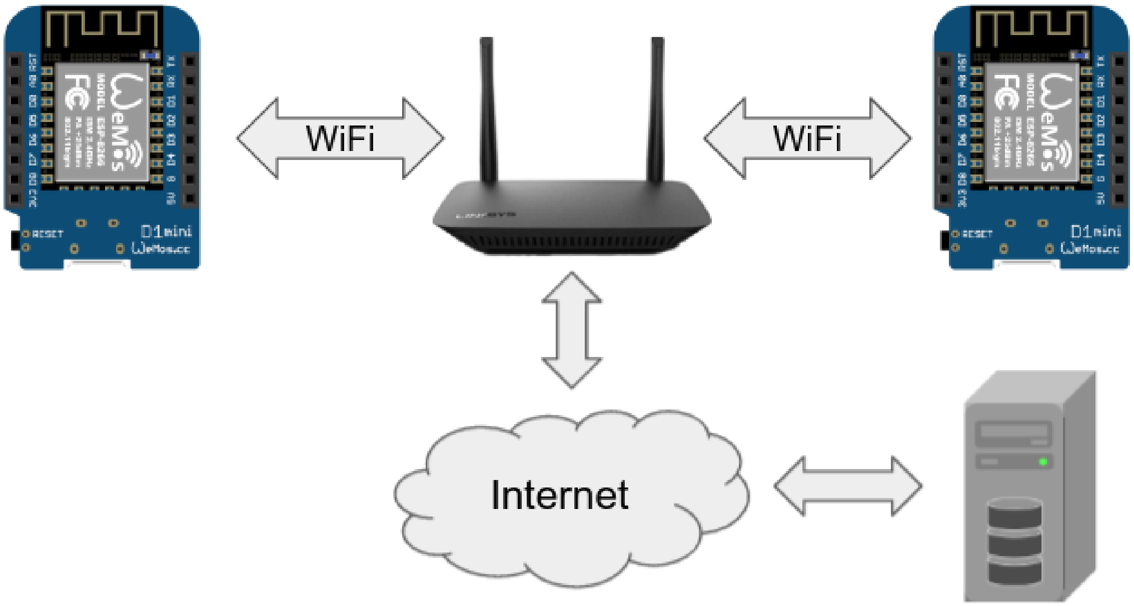
Other architectures (connection options) are also possible, including scenarios without an Internet connection. For example, the following figure shows the possibility of two ESP8266 boards exchanging data directly with each other. In this situation, one board must be configured in access point mode, providing a WiFi network for connection, and another as a WiFi station. In addition, one card acts as an HTTP server and another as an HTTP client. This type of wireless network is called *ad-hoc.*



A similar architecture can be achieved using a wireless access point, which provides a WiFi network to connect the two boards. From the point of view of exchanging HTTP messages, the operation is the same as described above. This connection architecture is called infrastructure mode, as the wireless access point is considered a piece of infrastructure equipment.



In some network setups, the two boards may not communicate directly. Thus, it is also possible for two embedded systems to communicate through a server. Each of the boards illustrated in the figure below makes requests to a server. In addition, boards can publish or consume data from this server to leave messages that will be made available for other applications on other boards or devices to consult.



To perform the HTTP request, the client needs to know the uniform resource locator - *Uniform Resource Locator* (**URL**), which, in short, has the following structure:

http://address:port/path/resource?query_string

A real URL example has already been used in this book in the activity of activating a device via the Internet:

http://bipes.net.br/easymqtt/publish.php?session=4tzuu7&topic=rele&value=1

Detailing each part of the URL:

Part	Description
http://	Protocol: HTTP or HTTPS (secure, with encrypted transmission) In the example: **http://**
address	Address of the server on the Internet. This could, for example, be in the format of IP address or name, such as 34.95.149.23 or bipes.net.br In the example: **bipes.net.br**
:port	The port is an optional parameter, which is typically omitted. When omitted, the port is understood to be 80, the default port for web services. Using other port values is useful when more than one web server is associated with the same IP / address or when some *firewall* or security system blocks port 80. In the example, the port is omitted.
/path/	The path to the resource on the server. In the example: **/easymqtt/**
resource	The name of the resource to be used/accessed on that server. In the example: **publish.php**
?query_string	The **query_string** allows you to send parameters to the resource. Multiple parameters can be included in the **query_string** separated by the symbol **&**. In the example: **?session=4tzuu7&topic=rele&value=1**

HTTP Client

Accessing web services via HTTP is not just about content. The HTTP protocol has become a standard way to access thousands of services and send and receive data between devices. Here, an interesting concept is M2M or *Machine to Machine*, which means direct communication between machines. The URL to access a service in web services is also called **endpoint**.

Knowing the URL of the web service that will be used, a web client can perform HTTP requests to access content and perform actions associated with that web service. The most common HTTP clients are web browsers. Thus, it is possible to use Google Chrome or Firefox to access and test web service URLs discussed here.

In this sense, free and paid web services can be used via HTTP, made available in a format called *Application Programming Interface* (API). Thus, there are thousands of service options accessible for HTTP clients, such as consulting the weather forecast, carrying out financial operations, obtaining currency and stock quotes, making payments, sending messages via SMS, posting messages via Twitter, sending messages via Telegram, performing geolocation queries, controlling devices remotely, accessing sensor data, among others. In addition, the website https://any-api.com/ lists over 1400 APIs for possible integration into other applications.

HTTP Client: Sending SMS Messages

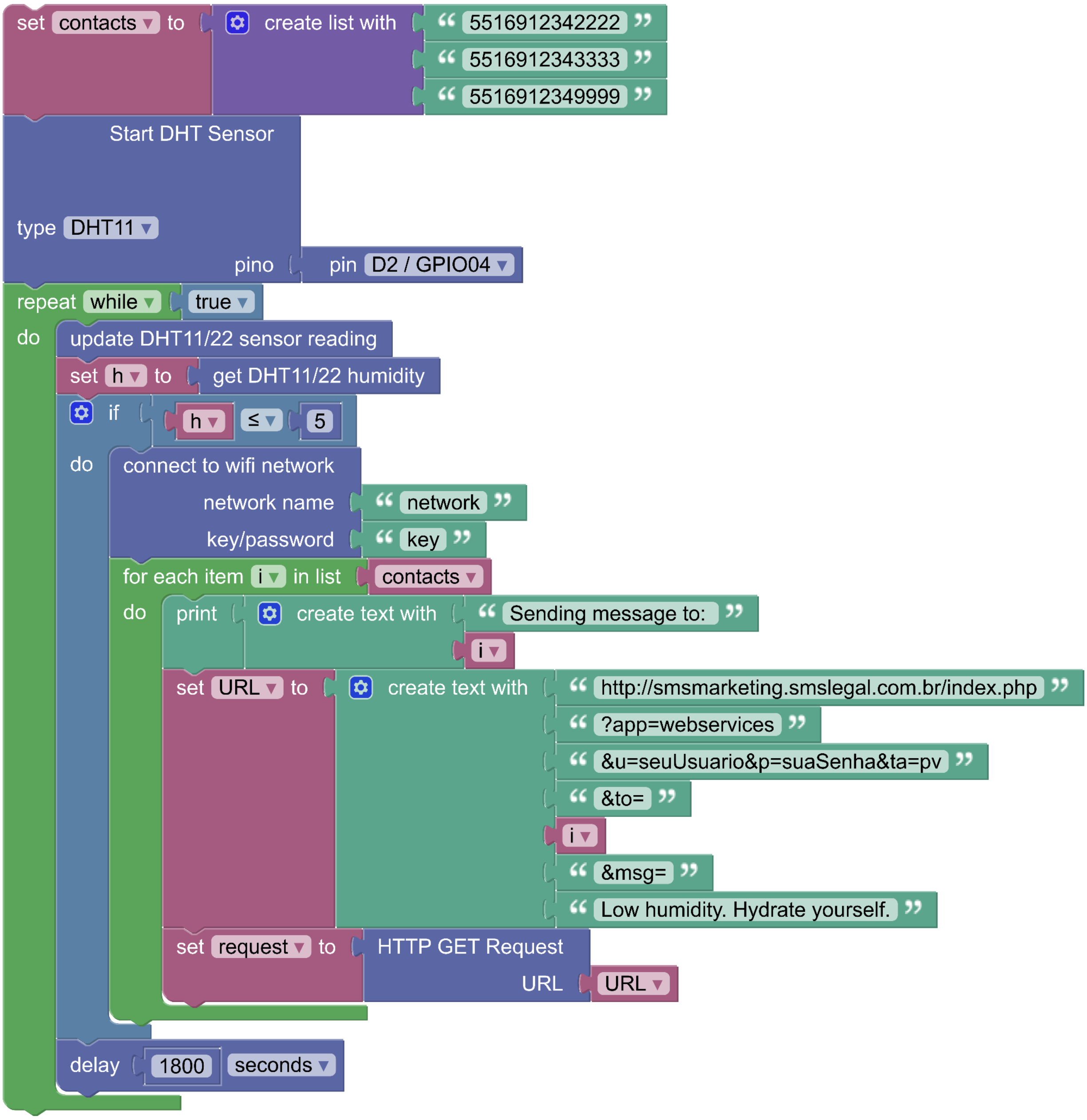
The following figure shows an example of a program that sends SMS messages to a list of phones, warning of low humidity when a DHT11 sensor measures relative humidity of less than 5%. The proposed system does not have any additional cell phone electronic module or SIM Card connected to the ESP8266 board to send SMS messages. Instead, the sending of the SMS message occurs through a web service provided by the following HTTP resource (**endpoint**): http://smsmarketing.smslegal.com.br/index.php. When accessing this resource, it is possible to send parameters (such as destination phone and message) through the query_string of the URL. For example:

http://smsmarketing.smslegal.com.br/index.php?app=webservices&u=seuUsuario&p=suaSenha&ta=pv&to=5516997970000&msg=BIPES

The above URL sends a message with “BIPES” to the phone number 5516997970000. This web service is paid, so both this and other web services require authentication. For this reason, **u** and **p** parameters can be used to inform username and password.

The example just shown is focused on Brazilian mobile phones. A quick search on https://any-api.com/ for “SMS” results on several SMS APIs. The first one found, for example, is “zoomconnect”, which provides the API **endpoint** https://www.zoomconnect.com/app/api/rest/v1/sms/send to send SMS to a large number of countries at low cost.

Anyway, as said, the next page shows an example of using an HTTP request to send SMS text messages to several phones:



```
set contacts to  create list with  "5516912342222"
                                   "5516912343333"
                                   "5516912349999"
Start DHT Sensor
type DHT11
                        pino  pin D2 / GPIO04
repeat while  true
do  update DHT11/22 sensor reading
    set h to  get DHT11/22 humidity
    if  h ≤ 5
    do  connect to wifi network
                      network name  "network"
                      key/password  "key"
        for each item i in list  contacts
        do  print  create text with  "Sending message to: "
                                     i
            set URL to  create text with  "http://smsmarketing.smslegal.com.br/index.php"
                                          "?app=webservices"
                                          "&u=seuUsuario&p=suaSenha&ta=pv"
                                          "&to="
                                          i
                                          "&msg="
                                          "Low humidity. Hydrate yourself."
            set request to  HTTP GET Request
                                        URL  URL
    delay  1800  seconds
```

HTTP Client: Changing color weather rooster

Perhaps you may remember those “*Changing Color Portuguese Weather Rooster*” gifts that change color according to weather conditions. We can build an IoT version of this rooster using an ESP board and a three-color RGB LED (*Red*, *Green,* and *Blue*). Climate information can be obtained from several sources. One example is the Brazilian National Institute for Space Research of the Ministry of Science, Technology, and Innovation (INPE/MCTI), which offers a web service for consulting climate information. Furthermore, it is possible to connect the ESP board to a *power bank*, turning this IoT rooster into a functional decoration item.

Considered a reference in space research, INPE/MCTI is Brazil's primary climate information and weather forecast source. In addition, INPE/MCTI's Center for Weather Forecasting and Climate Studies (CPTEC) offers a free web service for consulting climate information and weather forecasts for several Brazilian cities. More information about the service and its terms of use is available at: http://servicos.cptec.inpe.br/XML/ (information in Portuguese language only, sorry).

Anyway, similar services are available in most countries. In the USA, for example, the **National Weather Service** also provides a web service to query weather data. Information about this web service, its **endpoints,** and how to use them can be obtained on this link: https://www.weather.gov/documentation/services-web-api#.

A third and broader possibility is Open Weather Map (http://openweathermap.org/), which offers several web services to obtain current weather and forecasts with free and paid options.

Using INPE/MCTI data (http://servicos.cptec.inpe.br)

For the Brazilian (INPE/MCTI) web service, the first step to use this service is to verify the code (ID) of the city for which you want to obtain information. For this purpose, the web service http://servicos.cptec.inpe.br/XML/listaCidades offers a list of cities covered by the service and their respective codes. This list is provided in XML (formate*Xtensible Markup Language*), an international standard for exchanging information between machines and web services.

For this example, we will use the example of the city of São Carlos - SP, a Brazilian pole of science and technology[3]. The code for the city of São Carlos - SP is 4774. Thus, it is possible to use the previsao.xml resource: http://servicos.cptec.inpe.br/XML/cidade/4774/previsao.xml. Access to this **endpoint** generates the following XML result:

```
<cidade>
  <nome>São Carlos</nome>
  <uf>SP</uf>
  <atualizacao>2022-12-22</atualizacao>
  <previsao>
    <dia>2022-12-22</dia>
    <tempo>ci</tempo>
    <maxima>29</maxima>
    <minima>19</minima>
    <iuv>13.0</iuv>
  </previsao>
  <previsao>
    <dia>2022-12-23</dia>
    <tempo>ci</tempo>
    <maxima>27</maxima>
```

---

[3] Learn more about São Carlos - SP at https://www.reportsancahub.com.br/

```
    <minima>19</minima>
    <iuv>13.0</iuv>
  </previsao>
  <previsao>
    <dia>2022-12-24</dia>
    <tempo>c</tempo>
    <maxima>26</maxima>
    <minima>18</minima>
    <iuv>14.0</iuv>
  </previsao>
  <previsao>
    <dia>2022-12-25</dia>
    <tempo>ci</tempo>
    <maxima>24</maxima>
    <minima>18</minima>
    <iuv>14.0</iuv>
  </previsao>
</cidade>
```

Source: CPTEC/INPE (Access on Dec/2021)

The field **tempo** offers the information necessary to implement our application among the available information. ***Tempo*** is the Portuguese word for ***time***, but it also means ***weather***. Be happy! This book even teaches a bit of Portuguese.

In XML format, each piece of information is enclosed in special markers. For example, weather information is always delimited between <tempo> and </tempo>. In the example above, the most recent data reads: **`<tempo>ci</tempo>`**. The documentation for this INPE/MCTI web service also describes the possibilities of weather information. The table below shows some of these acronyms. See the link http://servicos.cptec.inpe.br/XML/ for a complete list of acronyms and descriptions.

Code	Description
ci	Isolated Showers
c	Rain
pc	Rain Showers
t	Storm
cl	Clear sky

Using Open Weather Map data (http://openweathermap.org/)

OpenWeatherMap has several APIs and **endpoints** with hundreds of possibilities. We are interested in the API that returns the current weather based on a query providing the city's name we want information about. More information, usage, and documentation about this **endpoint** can be found here: **https://openweathermap.org/current#name**. OpenWeatherMap allows up to 60 API calls per minute and 1,000,000 API calls per month on the free version. To use OpenWeatherMap APIs, you need to create an account and generate an API Key to authorize API requests: https://home.openweathermap.org/api_keys.

The **endpoint** of our interest is:

http://api.openweathermap.org/data/2.5/weather?q=Marrakesh&appid=KEY

Note that the above URL includes a parameter q=Marrakesh, where we specify that we want information for the city of Marrakesh. Moreover, you must provide an API key after appid=. Simply generate an API key and replace it with the word KEY. The API key will be something similar to `1b1111124567fafabcdee3aaae02dce0.`

Here are some examples of using such API:

API URL: **http://api.openweathermap.org/data/2.5/weather?q=Miami&appid=KEY**

Result for the city of **Miami**:

```
{"coord":{"lon":-80.1937,"lat":25.7743},"weather":[{"id":801,"main":"Clouds",
"description":"few clouds",
"icon":"02n"}],"base":"stations","main":{"temp":295.62,"feels_like":296.01,"temp_min":294.
15,"temp_max":297.16,"pressure":1015,"humidity":80},"visibility":10000,"wind":{"speed":0.
45,"deg":225,"gust":1.34},"clouds":{"all":20},"dt":1641429138,"sys":{"type":2,"id":2009435,"
country":"US","sunrise":1641384498,"sunset":1641422623},"timezone":-18000,"id":416413
8,"name":"Miami","cod":200}
```

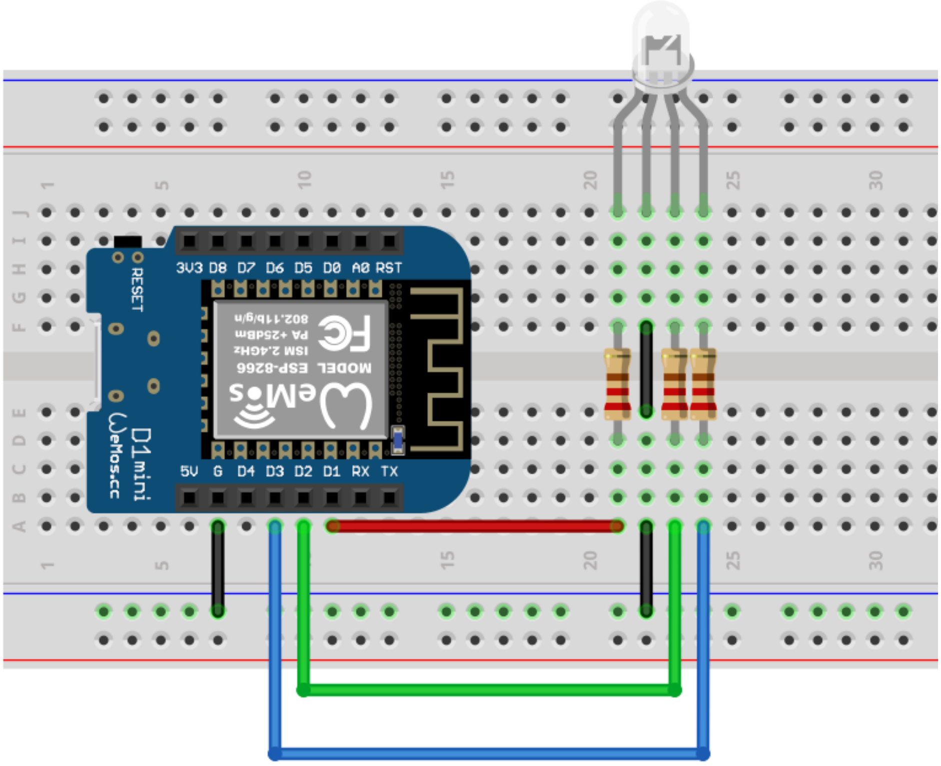
API URL: **http://api.openweathermap.org/data/2.5/weather?q=São Carlos&appid=KEY**

Result for the city of **São Carlos**:

```
{"coord":{"lon":-47.8908,"lat":-22.0175},"weather":[{"id":500,"main":"Rain",
"description":"light rain",
"icon":"10n"}],"base":"stations","main":{"temp":295.61,"feels_like":296.34,"temp_min":295.
61,"temp_max":295.61,"pressure":1008,"humidity":93,"sea_level":1008,"grnd_level":915},"
visibility":10000,"wind":{"speed":1.48,"deg":348,"gust":1.53},"rain":{"1h":0.15},"clouds":{"all
":100},"dt":1641429064,"sys":{"type":1,"id":8462,"country":"BR","sunrise":1641371643,"sun
set":1641419968},"timezone":-10800,"id":3449319,"name":"São Carlos","cod":200}
```

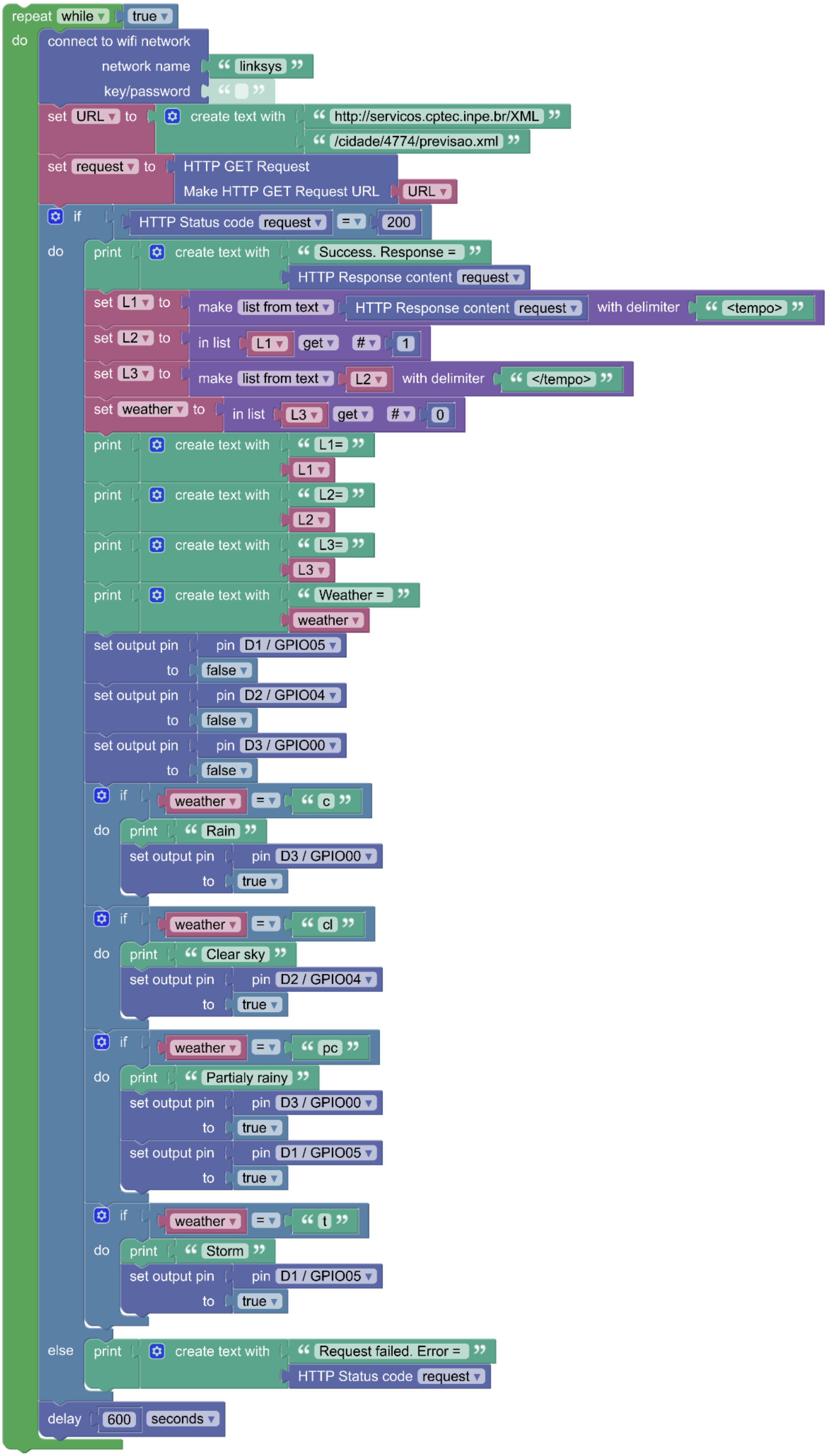
Note on the two examples that the field “description” has the information we need: “few clouds”, and “light rain”.

Based on the information provided by the mentioned APIs, it is possible to implement the "Changing color weather rooster". The following figure illustrates a possible assembly with a three-color LED connected to the ESP module. This LED has a ground terminal (GND) and three more terminals, one for each color. By controlling pins related to each LED color, it is possible to light the LED in red, green, blue, or combinations of these colors.



For the INPE/MCTI service: The presented program has an infinite repeat loop with a 10-minute wait on the next page. Thus, the system consults the INPE/MCTI server every 10 minutes and updates the LED colors according to the weather conditions for the city of **São Carlos**. Right after performing the **HTTP GET** request to the server, the program checks if the server responded with code 200, which means that the request was received, processed successfully, and the response is available. In this case, the answer is a text in XML format, as illustrated above.

The program then performs a sequence of operations on variables L1, L2, L3, and weather, to get just the abbreviation with information about the weather. Next, several **IF** conditional tests are performed to check some conditions and turn on the related LEDs: red (pin D1) for a storm, green (pin D2) for clear sky, blue (pin D3) for rain. A possibility was also implemented for the acronym pc (partially rainy), which lights up the LEDs connected to pins D1 and D3 simultaneously (red and blue), resulting in a color similar to purple.



```
repeat while true
do  connect to wifi network
        network name "linksys"
        key/password ""
    set URL to create text with
        "http://servicos.cptec.inpe.br/XML"
        "/cidade/4774/previsao.xml"
    set request to HTTP GET Request
        Make HTTP GET Request URL URL
    if HTTP Status code request = 200
    do  print create text with
            "Success. Response = "
            HTTP Response content request
        set L1 to make list from text HTTP Response content request with delimiter "<tempo>"
        set L2 to in list L1 get # 1
        set L3 to make list from text L2 with delimiter "</tempo>"
        set weather to in list L3 get # 0
        print create text with
            "L1="
            L1
        print create text with
            "L2="
            L2
        print create text with
            "L3="
            L3
        print create text with
            "Weather = "
            weather
        set output pin  pin D1 / GPIO05
                        to false
        set output pin  pin D2 / GPIO04
                        to false
        set output pin  pin D3 / GPIO00
                        to false
        if weather = "c"
        do  print "Rain"
            set output pin  pin D3 / GPIO00
                            to true
        if weather = "cl"
        do  print "Clear sky"
            set output pin  pin D2 / GPIO04
                            to true
        if weather = "pc"
        do  print "Partialy rainy"
            set output pin  pin D3 / GPIO00
                            to true
            set output pin  pin D1 / GPIO05
                            to true
        if weather = "t"
        do  print "Storm"
            set output pin  pin D1 / GPIO05
                            to true
    else print create text with
            "Request failed. Error = "
            HTTP Status code request
    delay 600 seconds
```

The listing below shows an example of the **Console** output when running the program. Note that printing the values of variables L1, L2, L3, and weather would not be necessary for the production version. Instead, we added these prints to show each step of obtaining the desired information through the list manipulation blocks. Some excerpts have been highlighted in bold to facilitate understanding of the messages below.

```
Waiting for Wifi connection
Connected

Success. Response = b"<?xml version='1.0'
encoding='ISO-8859-1'?><cidade><nome>São
Carlos</nome><uf>SP</uf><atualizacao>2021-12-22</atualizacao><previsao><dia>202
1-12-22</dia><tempo>c</tempo><maxima>29</maxima><minima>19</minima><iuv>13
.0</iuv></previsao><previsao><dia>2021-12-23</dia><tempo>ci</tempo><maxima>27</
maxima><minima>19</minima><iuv>13.0</iuv></previsao><previsao><dia>2021-12-24</
dia><tempo>c</tempo><maxima>26</maxima><minima>18</minima><iuv>14.0</iuv></p
revisao><previsao><dia>2021-12-25</dia><tempo>ci</tempo><maxima>24</maxima><
minima>18</minima><iuv>14.0</iuv></previsao></cidade>"

L1=['b"<?xml version=\'1.0\' encoding=\'ISO-8859-1\'?><cidade><nome>São
Carlos</nome><uf>SP</uf><atualizacao>2021-12-22</atualizacao><previsao><dia>2021
-12-22</dia>',
'c</tempo><maxima>29</maxima><minima>19</minima><iuv>13.0</iuv></previsao><pr
evisao><dia>2021-12-23</dia>',
'ci</tempo><maxima>27</maxima><minima>19</minima><iuv>13.0</iuv></previsao><pr
evisao><dia>2021-12-24</dia>',
'c</tempo><maxima>26</maxima><minima>18</minima><iuv>14.0</iuv></previsao><pr
evisao><dia>2021-12-25</dia>',
'ci</tempo><maxima>24</maxima><minima>18</minima><iuv>14.0</iuv></previsao></ci
dade>"']

L2=c</tempo><maxima>29</maxima><minima>19</minima><iuv>13.0</iuv></previsao>
<previsao><dia>2021-12-23</dia>

L3=['c',
'<maxima>29</maxima><minima>19</minima><iuv>13.0</iuv></previsao><previsao><di
a>2021-12-23</dia>']

Weather = c
Rain
```

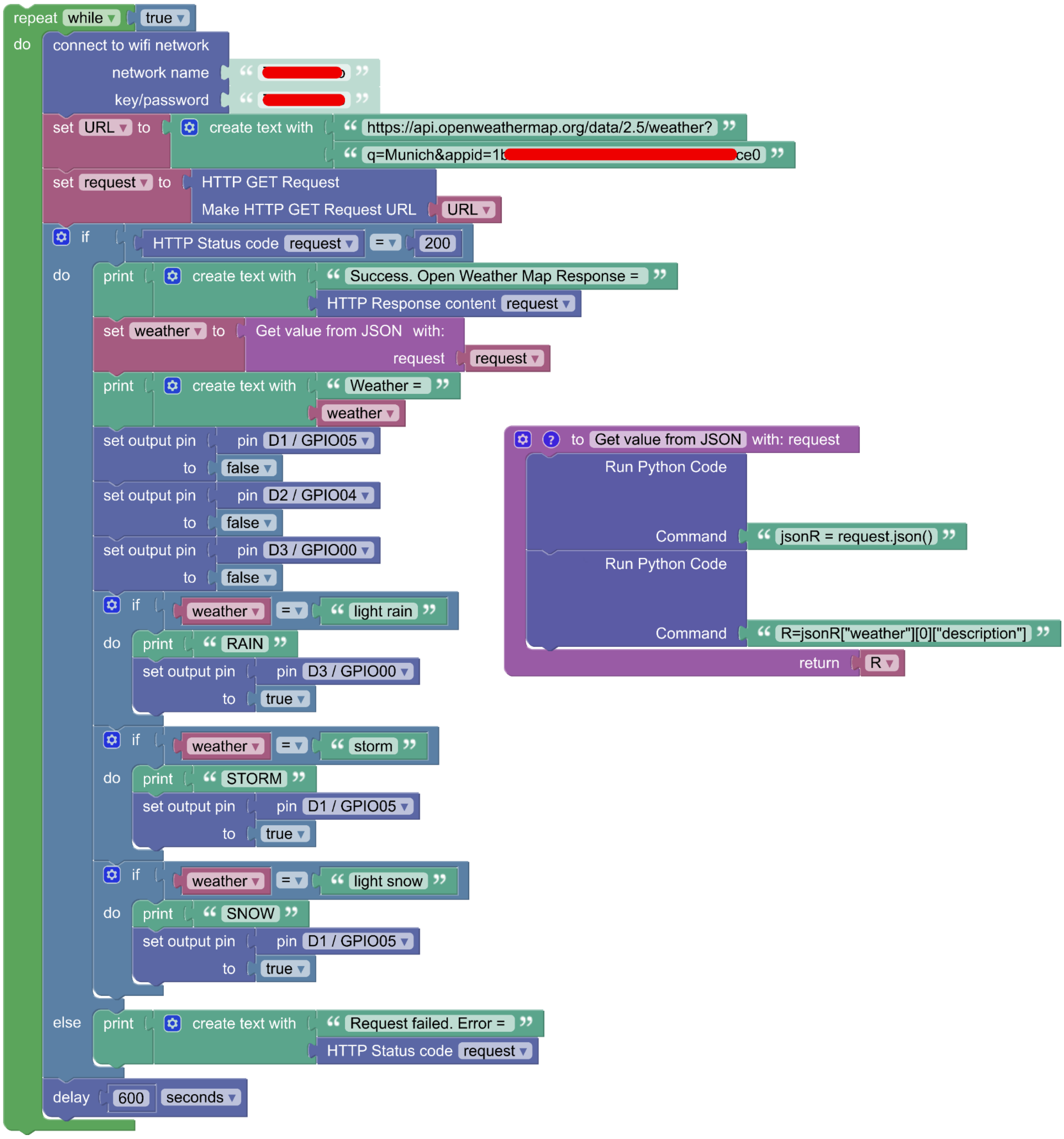
The following program shows a program using OpenWeatherMap API to query weather conditions at **Munich**. Note that OpenWeatherMap returns a JSON (JavaScript Object Notation), which should be handled differently from the XML. Moreover, we have hidden the API key value and network name and key with a red pen mark.

This program also helps us discuss another BIPES feature: directly executing Python commands inside blocks.

Micropython has a function to handle data in JSON format automatically. However, we have not prepared a block for that in BIPES. Thus, in two simple Python statements, we can easily access any information on the JSON response:

variableJSON = request.json()
variableData = variableJSON[“item”][index][“subitem”].

In that way, we have created a function **Get value from JSON** that receives the request as a parameter and internally runs these two Python statements mentioned above and returns with the value of interest to the main program. It is snowing in Munich now!

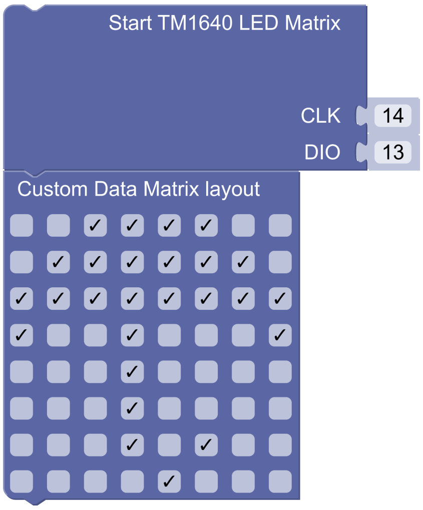


```
repeat while true
do  connect to wifi network
        network name "[redacted]"
        key/password "[redacted]"
    set URL to create text with "https://api.openweathermap.org/data/2.5/weather?"
                                "q=Munich&appid=1[redacted]ce0"
    set request to HTTP GET Request
                   Make HTTP GET Request URL URL
    if HTTP Status code request = 200
    do  print create text with "Success. Open Weather Map Response = "
                               HTTP Response content request
        set weather to Get value from JSON with:
                                      request request
        print create text with "Weather = "
                               weather
        set output pin  pin D1 / GPIO05
                        to false
        set output pin  pin D2 / GPIO04
                        to false
        set output pin  pin D3 / GPIO00
                        to false
        if weather = "light rain"
        do  print "RAIN"
            set output pin  pin D3 / GPIO00
                            to true
        if weather = "storm"
        do  print "STORM"
            set output pin  pin D1 / GPIO05
                            to true
        if weather = "light snow"
        do  print "SNOW"
            set output pin  pin D1 / GPIO05
                            to true
    else print create text with "Request failed. Error = "
                                HTTP Status code request
    delay 600 seconds

to Get value from JSON with: request
    Run Python Code
        Command "jsonR = request.json()"
    Run Python Code
        Command "R=jsonR["weather"][0]["description"]"
    return R
```

Finally, the following figure shows the system running on a Franzininho WiFi board (https://franzininho.com.br/), an open hardware board developed in Brazil and based on the ESP32S2 processor. The board has an RGB LED powered by a USB *power bank*, showing the current weather on its RGB LED, alongside a traditional “Weather Rooster.”

Optionally, it is even possible to include a LED matrix and present an umbrella or sun icon, depending on the weather conditions. In the example below, a matrix of LEDs with TM1640 controller was used, and each pixel of the matrix was defined in the “Custom Data Matrix Layout” block of the BIPES.

These LED matrices can be found in the *shield* format, an accessory that fits directly onto the ESP8266 board, avoiding the need for connection cables. In this case, the WeMos D1 mini board was used with the *shield* TM1640 LED Matrix. The following figure shows the blocks used to trigger the LED matrix and its result.



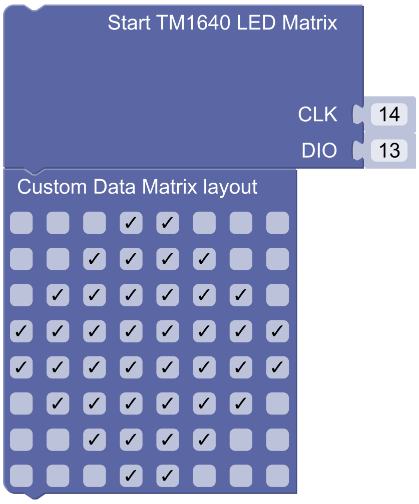
Start TM1640 LED Matrix

CLK 14

DIO 13

Custom Data Matrix layout

☐ ☐ ☑ ☑ ☑ ☑ ☐ ☐
☐ ☑ ☑ ☑ ☑ ☑ ☑ ☐
☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑
☑ ☐ ☐ ☑ ☐ ☐ ☐ ☑
☐ ☐ ☐ ☑ ☐ ☐ ☐ ☐
☐ ☐ ☐ ☑ ☐ ☐ ☐ ☐
☐ ☐ ☐ ☑ ☐ ☑ ☐ ☐
☐ ☐ ☐ ☐ ☑ ☐ ☐ ☐



Start TM1640 LED Matrix

CLK 14

DIO 13

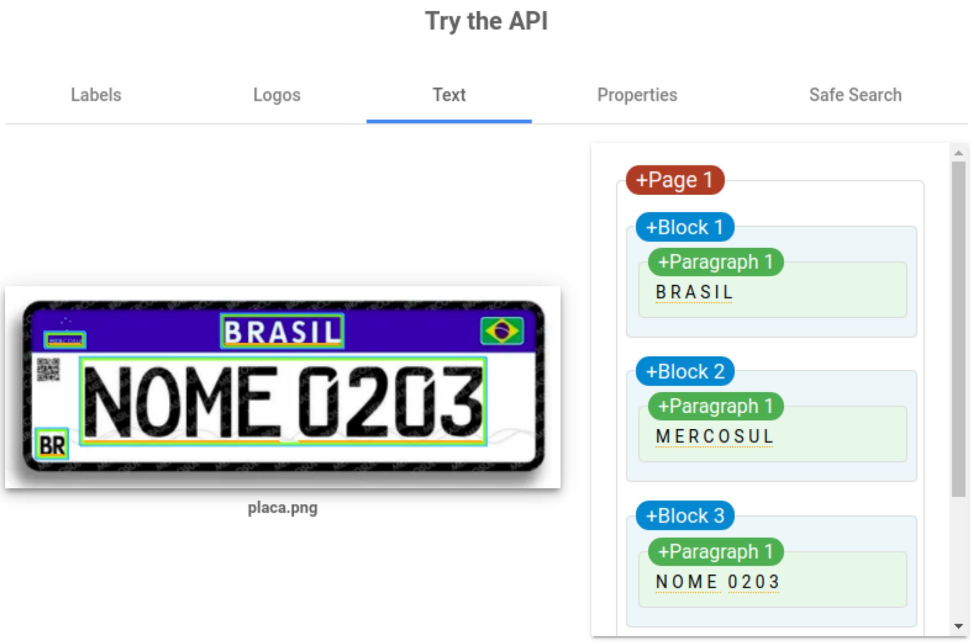
Custom Data Matrix layout

☐ ☐ ☐ ☑ ☑ ☐ ☐ ☐
☐ ☐ ☑ ☑ ☑ ☑ ☐ ☐
☐ ☑ ☑ ☑ ☑ ☑ ☑ ☐
☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑
☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑
☐ ☑ ☑ ☑ ☑ ☑ ☑ ☐
☐ ☐ ☑ ☑ ☑ ☑ ☐ ☐
☐ ☐ ☐ ☑ ☑ ☐ ☐ ☐

HTTP Client: Other Possibilities

As mentioned, there are thousands of API options for integration through HTTP requests. For example, we worked with the weather conditions API, but another helpful service includes sending messages via telegram through the **endpoint** https://api.telegram.org/bot<token>/sendMessage.

Another more advanced possibility is to add artificial intelligence and machine vision functionalities to an embedded system through the help of machine learning web services. Thus, even in an embedded system with limited computational resources, it is possible to use all the processing power of cloud services. An example of a possibility is in systems for analyzing images and identifying objects, people, and characters. Thus, a low-cost circuit, such as an ESP32-CAM module, can capture a photo and send it via HTTP to a cloud service. Google, Amazon, and IBM have advanced cloud image processing services, even infer whether a person is happy or sad, among other features. The following figure shows an example of a car license plate, which can be sent, via HTTP request, to the Google Cloud Vision API. As a result, the Google service returns a list of the characters present on the board. Note also that the system detected the phrase “MERCOSUL” present on the plate, in its upper left part, in a small dimension. More information on using this API can be found at: https://cloud.google.com/vision/docs/drag-and-drop.



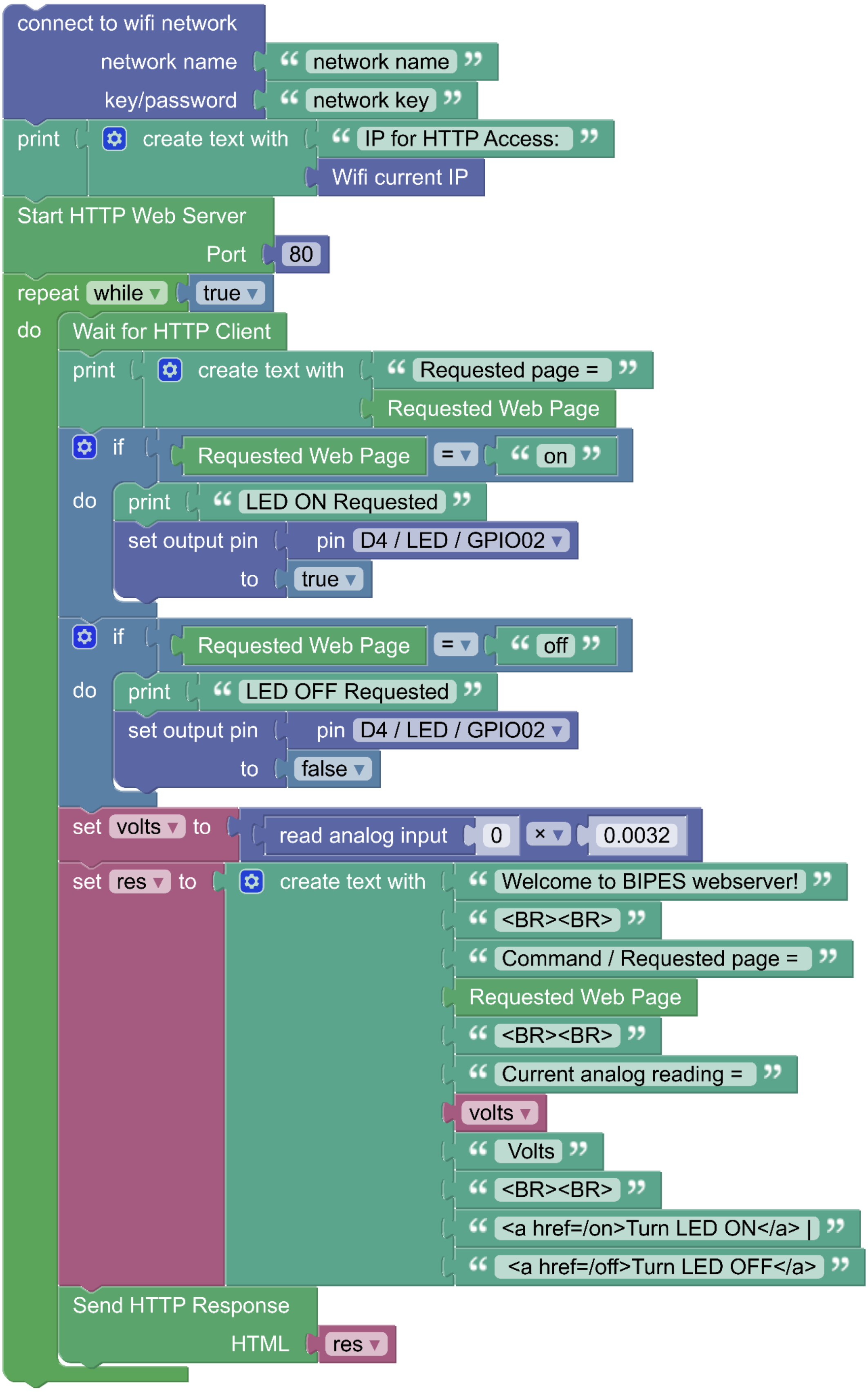
Note that this is an example of text recognition (also called OCR: optical character recognition), but the system could include object recognition, emotions, and other features in the image. You can also search the Internet for examples and projects using the ESP32-CAM module with the *Google Cloud Vision.* The embedded system developer community has already implemented and made available some options. These cloud features can also be helpful in robotics, automated optical inspection (AOI), and mechatronics applications. For more details on the use of integrated cloud with robots, a reading possibility is:

Aroca, RV, Péricles, A., de Oliveira, BS, Marcos, L., & Gonçalves, G. (2012, June). Towards smarter robots with smartphones. In the 5th workshop in applied robotics and automation, Robocontrol (pp. 1-6). sn.

HTTP Server

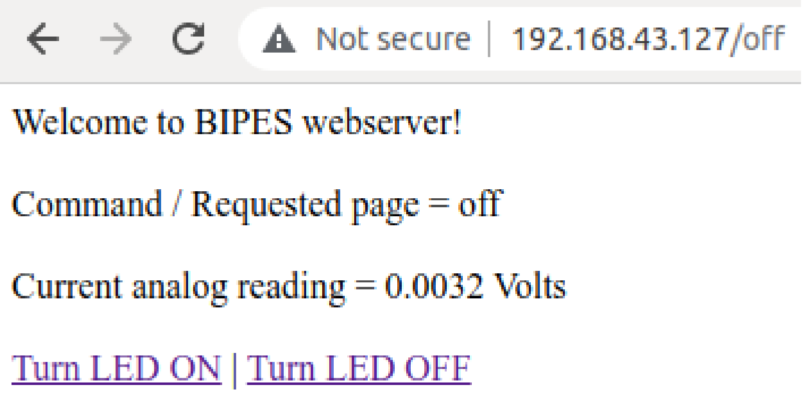
BIPES also allows an ESP board to act as an HTTP server, enabling direct access from any HTTP client, such as a web browser on a PC, a *smartphone*, or even another ESP32 or ESP8266 board using the block **HTTP GET Request.**

The following figure shows the implementation of an HTTP server in BIPES. The server is started on port 80 and continuously waits for HTTP requests from clients. When accessed by its IP address, the server displays the reading of the analog input of the ESP8266 board and offers two links to web pages on this server: **/on** and **/off.** Access to these pages is checked by the blocks "**IF Requested Web Page = on" (or off)** and generates a GPIO pin action to turn on/off the LED connected to pin D4. Using HTML codes, it is possible to insert figures, text boxes to request data from the user, among other possibilities for web / HTML development. The "Wifi Current IP" block was also used to check the IP address of your card, making it easier to identify the IP to access the card from other devices on your network.



```
connect to wifi network
    network name  "network name"
    key/password  "network key"
print  create text with  "IP for HTTP Access: "
                         Wifi current IP
Start HTTP Web Server
    Port  80
repeat while  true
do
    Wait for HTTP Client
    print  create text with  "Requested page = "
                             Requested Web Page
    if  Requested Web Page  =  "on"
    do  print  "LED ON Requested"
        set output pin  pin D4 / LED / GPIO02
                        to true
    if  Requested Web Page  =  "off"
    do  print  "LED OFF Requested"
        set output pin  pin D4 / LED / GPIO02
                        to false
    set volts to  read analog input 0  ×  0.0032
    set res to  create text with  "Welcome to BIPES webserver!"
                                  "<BR><BR>"
                                  "Command / Requested page = "
                                  Requested Web Page
                                  "<BR><BR>"
                                  "Current analog reading = "
                                  volts
                                  " Volts"
                                  "<BR><BR>"
                                  "<a href=/on>Turn LED ON</a> | "
                                  " <a href=/off>Turn LED OFF</a>"
    Send HTTP Response
        HTML  res
```

The result can be seen from a web browser, accessing the ESP board IP:



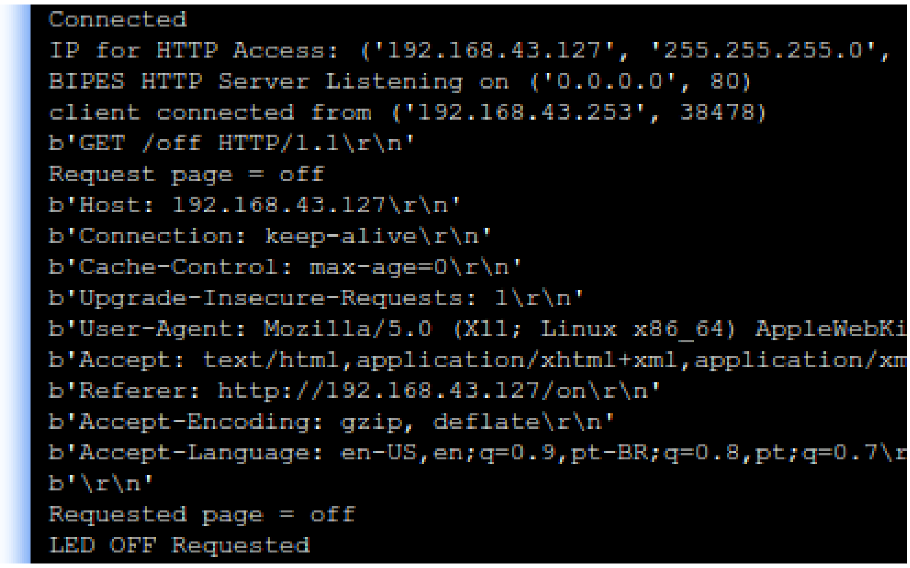
Welcome to BIPES webserver!

Command / Requested page = off

Current analog reading = 0.0032 Volts

Turn LED ON | Turn LED OFF

And, as mentioned, can be monitored by the **Console:**



```
Connected
IP for HTTP Access: ('192.168.43.127', '255.255.255.0',
BIPES HTTP Server Listening on ('0.0.0.0', 80)
client connected from ('192.168.43.253', 38478)
b'GET /off HTTP/1.1\r\n'
Request page = off
b'Host: 192.168.43.127\r\n'
b'Connection: keep-alive\r\n'
b'Cache-Control: max-age=0\r\n'
b'Upgrade-Insecure-Requests: 1\r\n'
b'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKi
b'Accept: text/html,application/xhtml+xml,application/xm
b'Referer: http://192.168.43.127/on\r\n'
b'Accept-Encoding: gzip, deflate\r\n'
b'Accept-Language: en-US,en;q=0.9,pt-BR;q=0.8,pt;q=0.7\r
b'\r\n'
Requested page = off
LED OFF Requested
```

In most cases, this webserver will only receive HTTP requests from devices connected on the same WiFi network that the ESP board is running the HTTP server application.

This limitation depends on the network architecture used: in most cases, the wireless access point or router used obtains a unique Internet (IP) address from the Internet Provider that is public and accessible to receive connections coming from the Internet. Such a router or access point distributes internal/private IP addresses that are not accessible from any other device on the Internet. These internal IPs are in the form 192.168.x.y, 172.x.y.z or 10.x.y.z. The single public IP is then shared with all internal devices through *Network Address Translation* (NAT). Thus, as the ESP module has a private IP, for it to be accessed from the Internet, it is necessary to configure a **port forwarding** strategy, which will be presented in the next section.

Anyway, even with the IP with access only on the internal private network, all devices on the same network can access the ESP board's web server. So the board's IP could be configured, for example, in an application made in MIT App Inventor, allowing to integrate *smartphone* applications with the ESP board through HTTP requests.

For more details about the NAT process and IP addressing, consult a specialized book on computer networks, such as:

TANENBAUM, AS – Computer Networks – 4th Ed., Editora Campus (Elsevier)

Integration with Google Home or Amazon Alexa

The Internet of Things has seen significant growth in home automation, also called *domotics*. In this context, two devices, also embedded systems, offer high quality and flexible personal assistant options: **Google Home** and **Amazon Alexa**. Among the various features of these devices, the main one is the interaction with the user through voice commands and responses, where the user can make requests by voice in a straightforward way. For example, some possible commands that can be used are: "Schedule an appointment", "Turn on the light", "Send a message to Matheus", "Remind me to get my clothes out of the washing machine in 10 minutes," or "Make a phone call to Lilian".

The flexibility of these personal assistants includes the possibility of integrating them with new services and applications and customizing actions. We will use the IFTTT service among the various integration possibilities: ***If This Then That*** (https://ifttt.com/), free for up to 5 automation actions, called *Applets*. In addition, IFTTT offers hundreds of event options (***If This***) that can trigger hundreds of other action options (***Than That***). In addition to hundreds of options, IFTTT is also compatible with Google Home and Amazon Alexa. And, in particular, IFTTT has the ***WebHook*** action that allows it to perform an HTTP request when some event is detected: that is, the HTTP request is completed in the context of the action (*Than That)*.

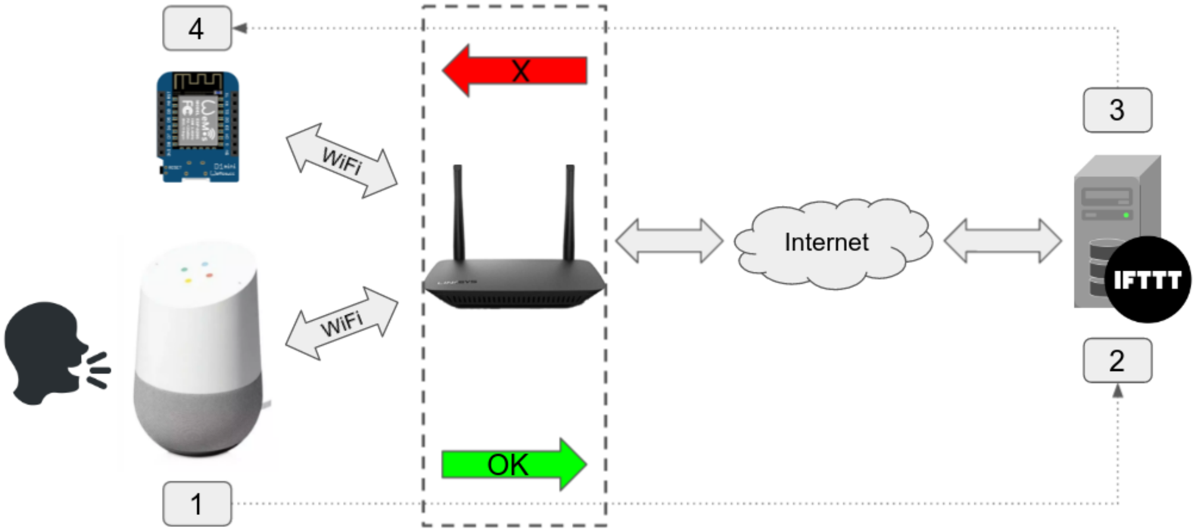
Thus, it is possible to create an IFTTT *Applet* that waits for a phrase like "Turn on the Light", and when this phrase is spoken, it performs an HTTP request to the BIPES web server embedded in the BIPES board itself. However, as already mentioned, the HTTP server embedded by BIPES on the ESP board will be inaccessible to the IFTTT service server, so it will be necessary to carry out a technical adjustment so that the request reaches the ESP board. Below are two options for the data from Google Home or Amazon Alexa, passing through the IFTTT, to reach the ESP board that controls the desired device.

Option 1: Port Forwarding

The following figure illustrates the architecture of a system where a voice command made through Google Home results in turning an electrical appliance on or off through an

ESP8266 board. In the Figure, the user speaks the voice command that we defined as "***Shine my light***".

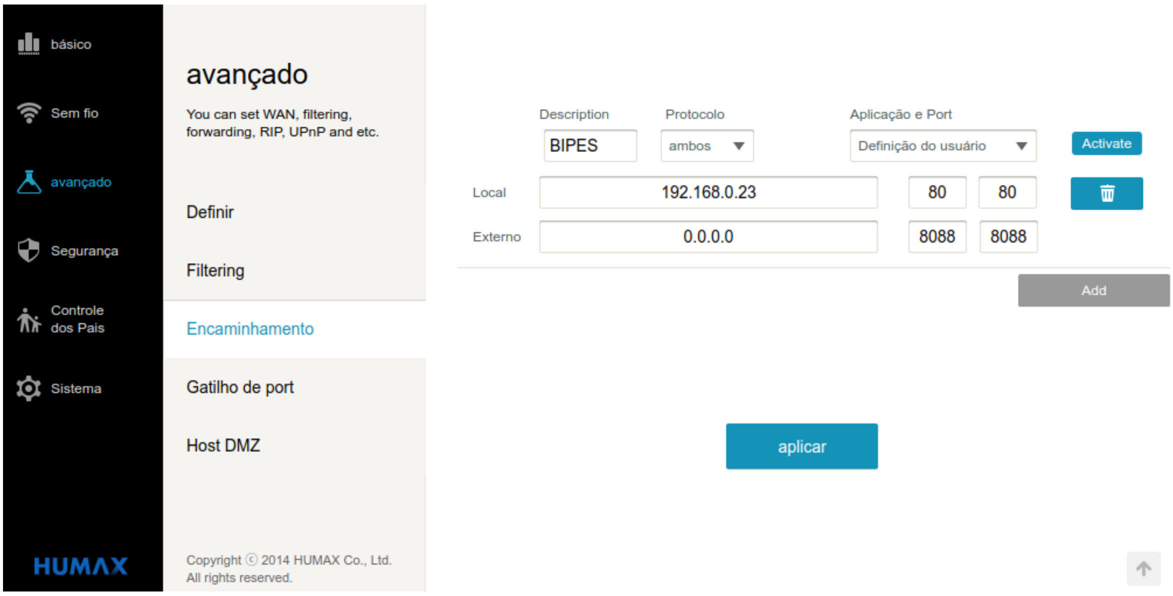
Thus, Google Home receives the voice command and sends (1) data to the IFTTT server (2), a cloud service, which processes the command in its *Applet* and performs an HTTP request (3) that is received by the ESP8266 board (4). Note that the data flow [(1) to (2)] has a green arrow, where, in most cases, there are no restrictions for sending requests from an internal private IP address to an external public IP address. However, the IFTTT server request to the ESP8266 board [(3) to (4)] may not reach the ESP8266. For that to happen, it is necessary to configure port forwarding in the router or access point or firewall (we mentioned several **or(s)**, as the architecture of each network might require different configurations).



For this configuration, it is necessary to configure a TCP port forwarding from your network's public/external IP to the internal IP of the ESP8266 board. First, check your network equipment manual and options. As an illustration, the following figure shows the port forwarding configuration on a HUMAX integrated cable modem/router.

According to the setup that we already performed in the previous section, an ESP8266 board received the private IP 192.168.0.23 and can be accessed, from the internal network, at the address http://192.168.0.23/. Note that the HUMAX router has been configured in Advanced Options → Forwarding, enabling a port forwarding from the external address 0.0.0.0 (any external address) to the internal address 192.168.0.23. Furthermore, it has also been configured to map the external port 8088 to the internal port 80 to the internal IP 192.168.0.23.

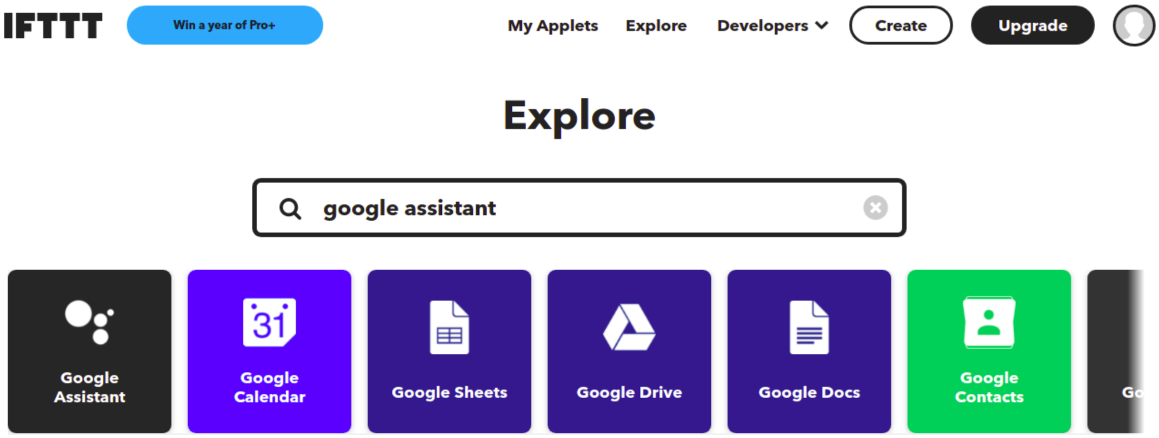
The next step is to check what is your public IP address. One possibility is to search Google for "*what is my IP address*". Google itself will display an IP address identified as "*Your public IP address*". With port forwarding active, you can access your IP from anywhere on the Internet, followed by port 8088. For example, if the IP is 187.66.80.187, use a web browser to go to the address http://187.66.80.187:8088/. Next, ensure the HTTP server program is active on the ESP8266 board. If you can access the board through this public IP address, IFTTT will also access it.



Description: BIPES
Protocolo: ambos
Aplicação e Port: Definição do usuário
Local: 192.168.0.23 80 80
Externo: 0.0.0.0 8088 8088

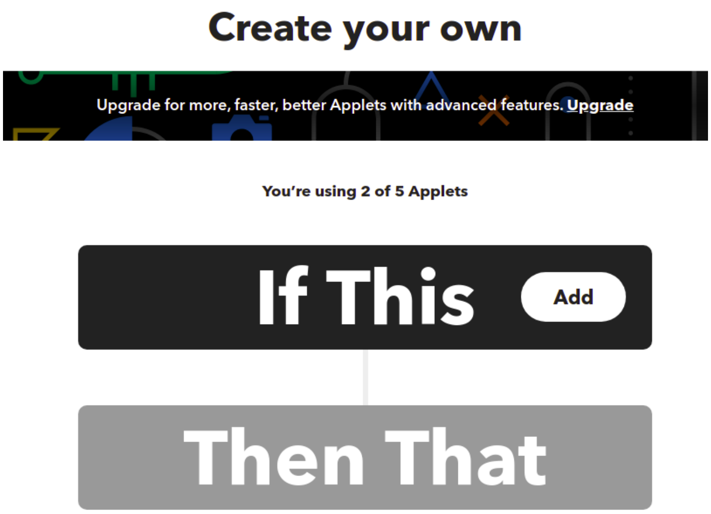
Before proceeding, a little note: your public IP address can dynamically change after a few hours, days, or weeks. This possibility of change may occur depending on your Internet Service Provider (ISP). Therefore, if you need to use the system frequently, it is interesting to configure a dynamic DNS service (https://www.noip.com/pt-BR), which will provide you with a name on the Internet, such as **myhome.noip.com**. This name will continuously be updated and synchronized with your public IP address, even if the IP changes.

Now that we know the public IP to perform HTTP requests, we can configure the IFTTT *Applet*. First, go to https://ifttt.com/ and create an account if you don't have one. It is important to use the same email/account already used with your Google Home or Alexa. A setup for Google Home is illustrated below, similar to Amazon Alexa. After creating the account, click ***Explore*** and look for *Google Assistant*, as shown:



google assistant

Follow the following steps and confirm the association of IFTTT with Google Home. This procedure only needs to be done once. Then it is possible to create an IFTTT *Applet*. To do so, click on ***Create*** on the IFTTT main screen:

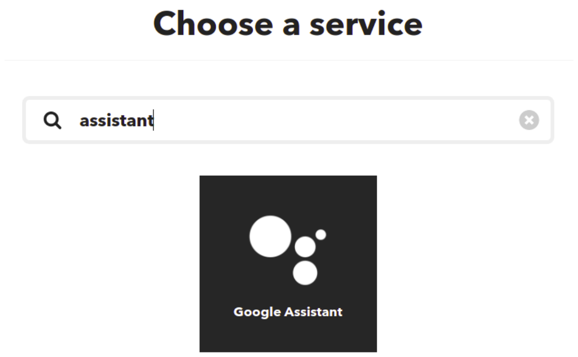


Create your own

Upgrade for more, faster, better Applets with advanced features. Upgrade

You're using 2 of 5 Applets

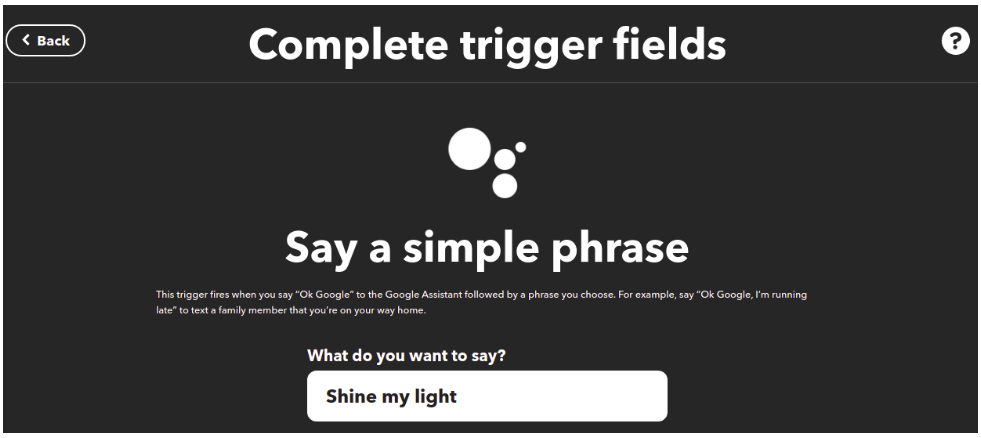
Click on "***If This***" and choose ***Google Assistant:***



Choose a service

assistant

Choose the option "***Say a simple phrase***" and enter a phrase. In this example, we use "***Shine my light***". It is also possible to specify what the system will say in response to this command through the field "***What do you want the Assistant to say in response?***".



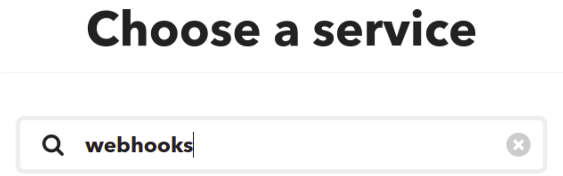
Complete trigger fields

Say a simple phrase

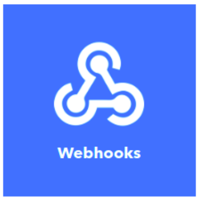
This trigger fires when you say "Ok Google" to the Google Assistant followed by a phrase you choose. For example, say "Ok Google, I'm running late" to text a family member that you're on your way home.

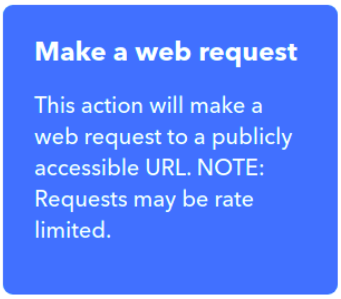
What do you want to say?: Shine my light

Click ***Create trigger***. Next, let's add the action to be done. Click on **"Then That"** and choose ***Webhook*** and then select the option "***Make a web request***".


Choose a service

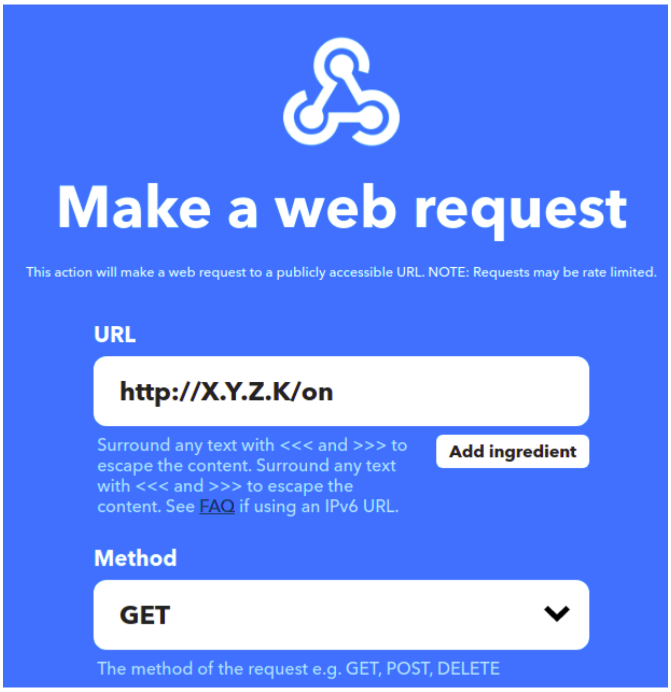
webhooks




Make a web request

This action will make a web request to a publicly accessible URL. NOTE: Requests may be rate limited.

In "**Make a web request**", enter the full **URL** of the HTTP service that reaches the ESP8266. The URL must include your public IP address with the resource path and name—for example, http://187.66.80.187:8088/on or http://187.66.80.187:8088/off. The figure below shows the URL http://XYZK/on, generically. Note that the correct public address must be entered in this field. Also, choose the GET method for the HTTP request to be performed.


Make a web request

This action will make a web request to a publicly accessible URL. NOTE: Requests may be rate limited.

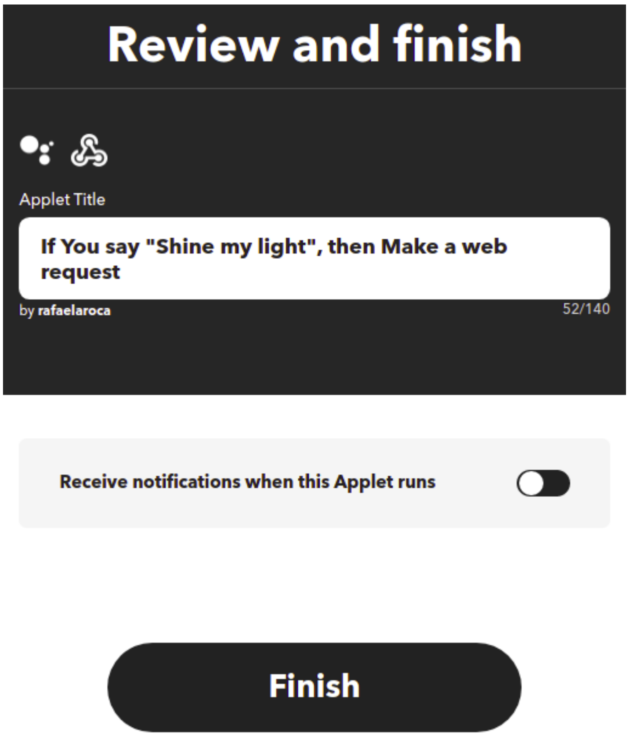
URL: http://X.Y.Z.K/on

Surround any text with <<< and >>> to escape the content. Surround any text with <<< and >>> to escape the content. See FAQ if using an IPv6 URL.

Method: GET

The method of the request e.g. GET, POST, DELETE

Finally, click ***Create action*** and then ***Continue***, followed by ***Finish*.** Eventually, to test it, go to Google Home and say: "***Ok Google, Shine my Light!***".



Review and finish

Applet Title

If You say "Shine my light", then Make a web request

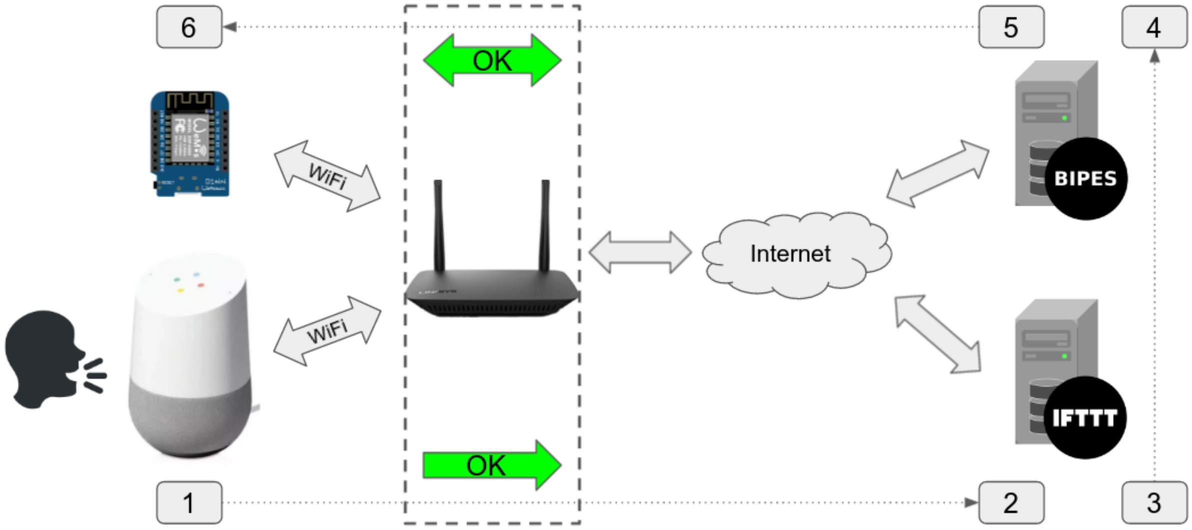
by rafaelaroca 52/140

Receive notifications when this Applet runs

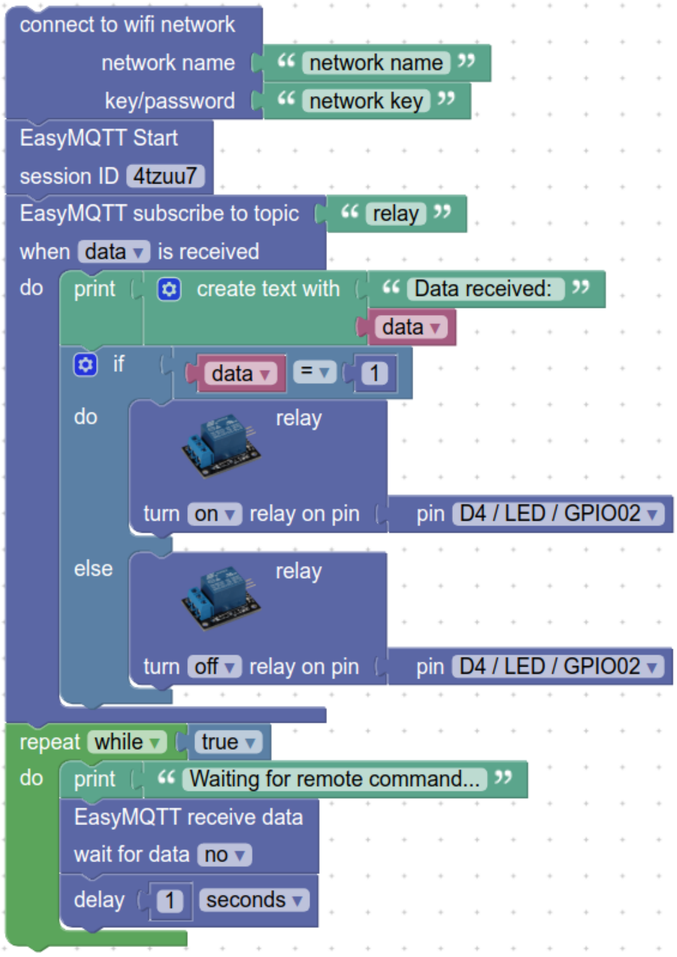
Option 2: Integration with IFTTT through EasyMQTT

It may occur that the router/access point or *firewall* password is not available, and it is not possible to configure port forwarding. Some ISPs may also end up not providing public IPs to their customers or, for security reasons, offer Internet connections with very restrictive *firewalls*, limiting the reception of Internet data streams that have not been initiated from their network. For these and other cases, it is possible to integrate BIPES and IFTTT through the webserver http://bipes.net.br of the BIPES project itself.

The following figure illustrates this system architecture: the user issues a voice command "***Shine my light***" to Google Home (1) that requests the IFTTT server (2). IFTTT then runs an *Applet* (3) that performs an HTTP GET request to the BIPES project's web server (http://bipes.net.br) (4), publishing a value in a topic of an EasyMQTT BIPES session. The BIPES server, in turn, updates that session (5) synchronously with the ESP8266 board, which must be running a program **subscribed** in an EasyMQTT topic, which then receives the data and performs an action (6).



Note that there is no HTTP server on the ESP8266 board this time, but only a program subscribed to a topic of an EasyMQTT session. This program has already been illustrated in the “**Controlling Devices via the Internet**” section. However, we present the same program here again for ease of understanding. This program subscribes to the relay topic of the EasyMQTTsession **4tzuu7** and, on any change in this topic, performs the action (block **“subscribe to the topic”).**

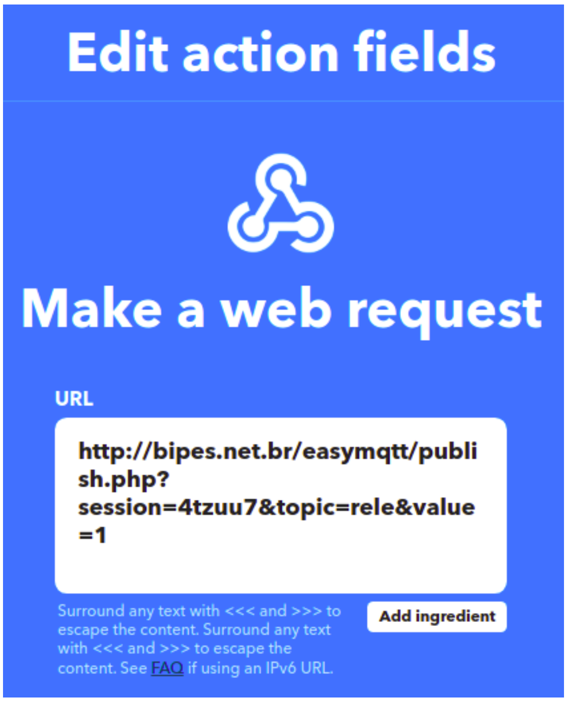


```
connect to wifi network
          network name  “ network name ”
          key/password  “ network key ”
EasyMQTT Start
session ID 4tzuu7
EasyMQTT subscribe to topic  “ relay ”
when data is received
do  print  create text with  “ Data received: ”
                             data
    if  data = 1
    do  relay
        turn on relay on pin  pin D4 / LED / GPIO02
    else  relay
        turn off relay on pin  pin D4 / LED / GPIO02
repeat while true
do  print  “ Waiting for remote command... ”
    EasyMQTT receive data
    wait for data no
    delay 1 seconds
```

Remember that we have already used a BIPES project server URL to change an EasyMQTT session and control a device from a *smartphone App* made with MIT App Inventor. The URL used for integration with a *smartphone* application was:

http://bipes.net.br/easymqtt/publish.php?session=4tzuu7&topic=relay&value=1

Now, this same URL can be used for the integration with IFTTT. The *Applet* IFTTTis identical to the previous one, only changing the URL triggered by the IFTTT when the event occurs:


Edit action fields

Make a web request

URL

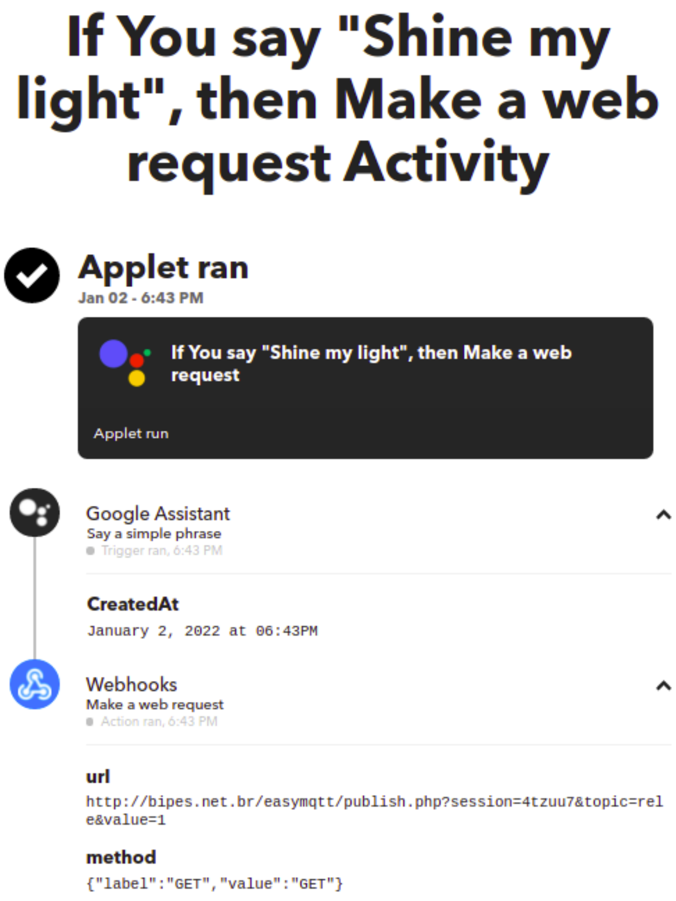
http://bipes.net.br/easymqtt/publish.php?session=4tzuu7&topic=rele&value=1

Surround any text with <<< and >>> to escape the content. Surround any text with <<< and >>> to escape the content. See FAQ if using an IPv6 URL.

Save the changes and test them by asking Google Home: "***Shine my light***". In this case, no port forwarding/packet forwarding configuration is needed, as in option 1. Hence, the system works independently of the network configuration and properties, but it depends on the BIPES server.

Note: remember that BIPES is an open and free project, and the session used as an example here (4tzuu7) may be used by other users/readers, causing unexpected effects (change in the state of a topic). So, define a unique session for your project. You can check if an EasyMQTT session is empty/available via the URL: https://bipes.net.br/beta2/easymqtt/getsession.php?session=12, where 12 is the desired session to be verified.

Finally, it may be helpful to know that the IFTTT offers the option "***View Activity***", which allows you to check the records (*logs*) of reception of events and the results. In particular, in the case of ***Webhooks***, it is possible to check if the command was sent to the server and the result of the HTTP request. The figure below shows an example of a request made to the EasyMQTT server of the BIPES project.



If You say "Shine my light", then Make a web request Activity

Applet ran

Jan 02 - 6:43 PM

If You say "Shine my light", then Make a web request

Applet run

Google Assistant

Say a simple phrase

Trigger ran, 6:43 PM

CreatedAt

`January 2, 2022 at 06:43PM`

Webhooks

Make a web request

Action ran, 6:43 PM

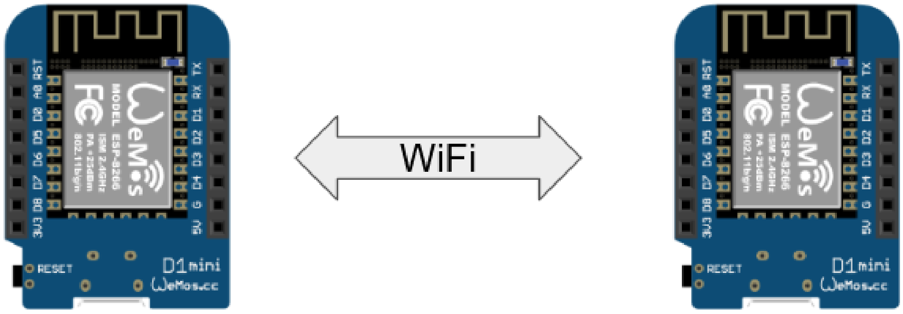
url

```
http://bipes.net.br/easymqtt/publish.php?session=4tzuu7&topic=rel
e&value=1
```

method

```
{"label":"GET","value":"GET"}
```

Activity: server and client - remote control via WiFi



Use a pair of ESP8266 boards, programming one to act as a wireless access point and HTTP server. Next, configure another as a WiFi station and client so that this second board, the client, sends an HTTP request every time a digital pin is triggered (by a sensor, a button, or other devices). When the board configured as a server receives the HTTP request, it should switch an output pin to turn on/off any device. In this way, an ESP8266 board will act as a wireless remote control via WiFi of a device connected to the ESP8266 card acting as a “server”.

* * * *

Congratulations!

You have developed complete Internet of Things (IoT) applications with cloud, sensors, remote control, and *dashboards* for visualization and control!

* * * *

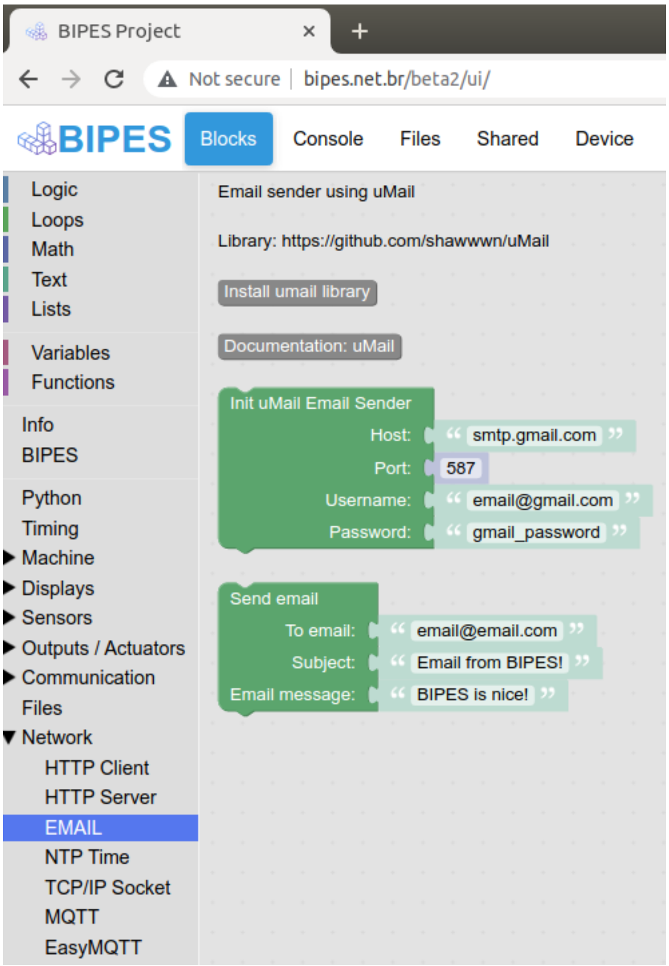
Extra activities

Sending email

Activity:
Design a program that sends an email to you when a digital input pin (GPIO) on the ESP8266 detects a change in your input signal (from 0 to 1, for example). This pin can be connected to a presence sensor or an open door sensor (*reed switch).* Upon detecting the change, the program must send the email.

Tips:

1. For this activity, the ESP8266 board must be connected to the Internet;
2. BIPES has a modular library system. For example, you need to install the uMail library to send emails. The installation can be quickly done by accessing Network -> EMAIL and then clicking on “**Install mail library**”.
    a.

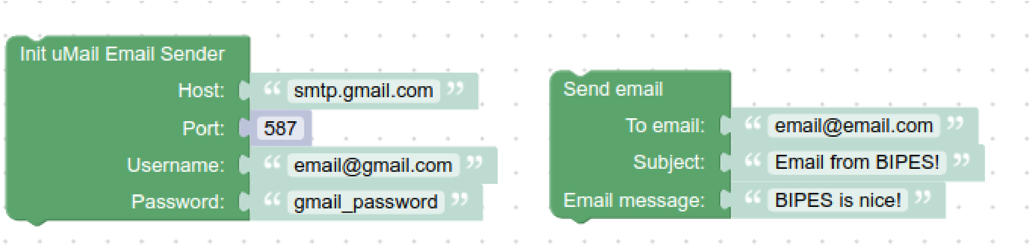


Email sender using uMail

Library: https://github.com/shawwwn/uMail

Init uMail Email Sender
Host: smtp.gmail.com
Port: 587
Username: email@gmail.com
Password: gmail_password

Send email
To email: email@email.com
Subject: Email from BIPES!
Email message: BIPES is nice!

3. After installing the library, you can use its launch blocks and email sending blocks:
   a. 

Init uMail Email Sender
Host: smtp.gmail.com
Port: 587
Username: email@gmail.com
Password: gmail_password

Send email
To email: email@email.com
Subject: Email from BIPES!
Email message: BIPES is nice!

GMAIL

BIPES uses MicroPython's umail library, which allows it to send emails using virtually any email provider. However, each provider may need specific settings.

Gmail, for example, requires you to create an ***application password*** so that ESP8266 can send emails directly. Thus, the GMAIL account must have 2-factor authentication enabled and the “App Password” enabled.

The following link has a further explanation:
https://myaccount.google.com/apppasswords

At this link, select:

App -> Email,
Device -> Other

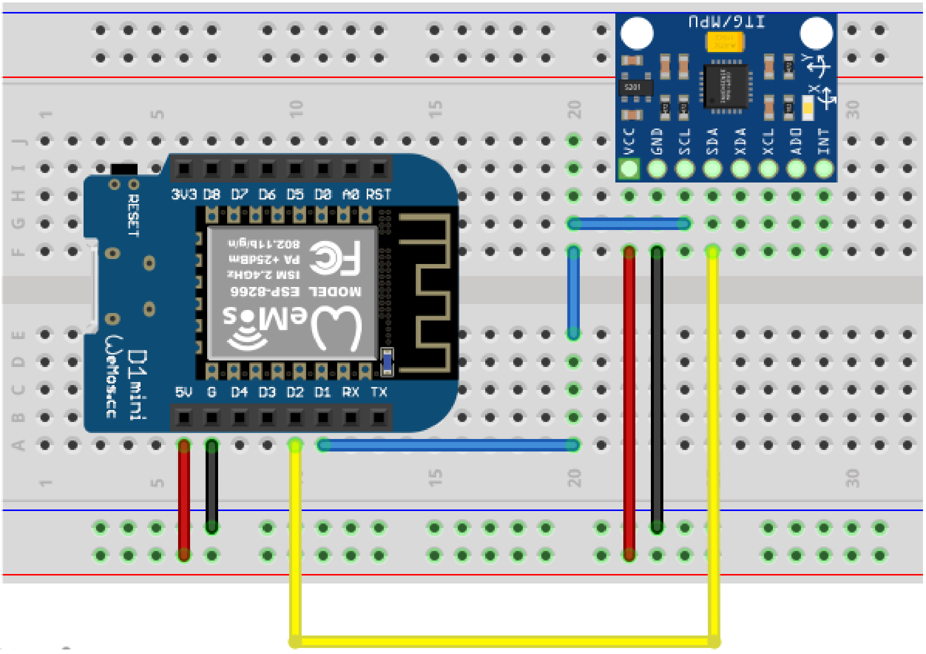
Gmail will generate a dedicated password for the ESP8266. Copy this password and paste it into the Init uMail block.

Inertial Measurement Unit - MPU6050

If you have an MPU6050 inertial measurement unit, you can use it according to the diagrams below and use the IMU / MPU6050 block:

The following table shows the connections:

MPU6050	**ESP8266pin**
GND	**GND**
VCC	**5V**
SCL	**5 (D1)**
SDA	**4 (D2)**



Activity:
Build an IoT dashboard that shows the variation of board orientation angles in real-time graphs.

Challenge:
Implement a program to calculate the angles (in this case, using accelerometers only) of inclination in the three axes from the gravity acceleration measurements and prepare an **arrow** on the IoT dashboard that indicates the tilt of the plate.

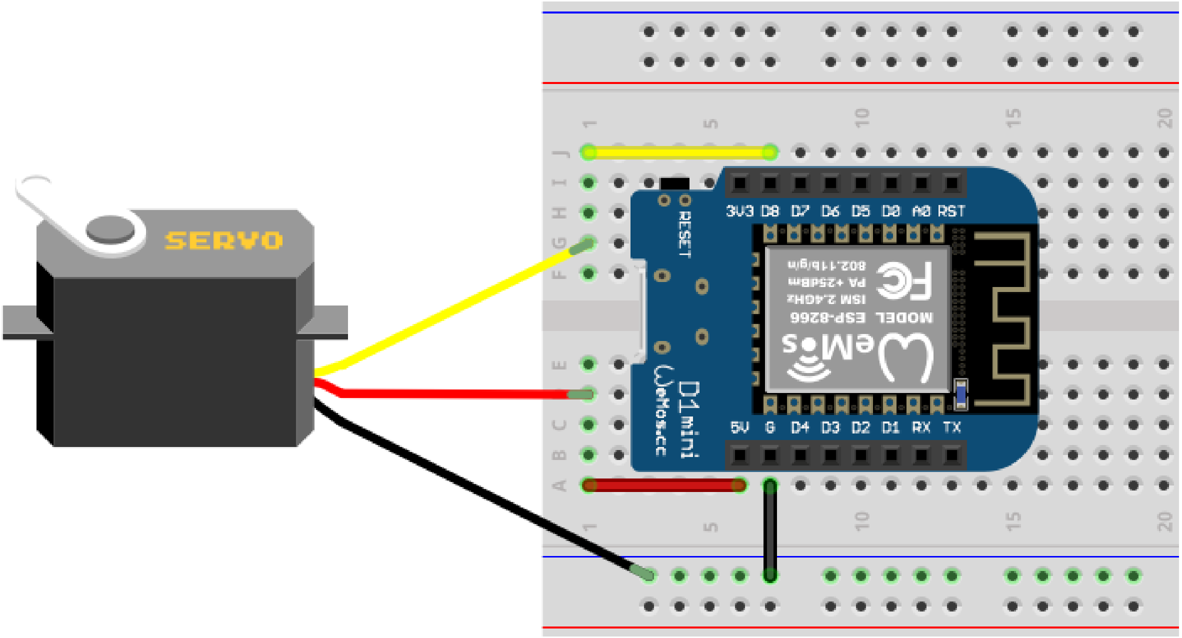
RC Servo Motor (for model aircraft)

Activity:
Develop a program with an IoT dashboard containing a **slider.** When the user changes the *slider* value, the servo should move to the angle set on the *slider*.

Tip: use the **RC Servo Motor** block on the left side toolbox **Outputs and Actuators**.

The following table shows the connections:

Servo	ESP8266Pin
GND (brown)	GND
VCC (red)	5V
S / Signal (orange)	15 (D8)

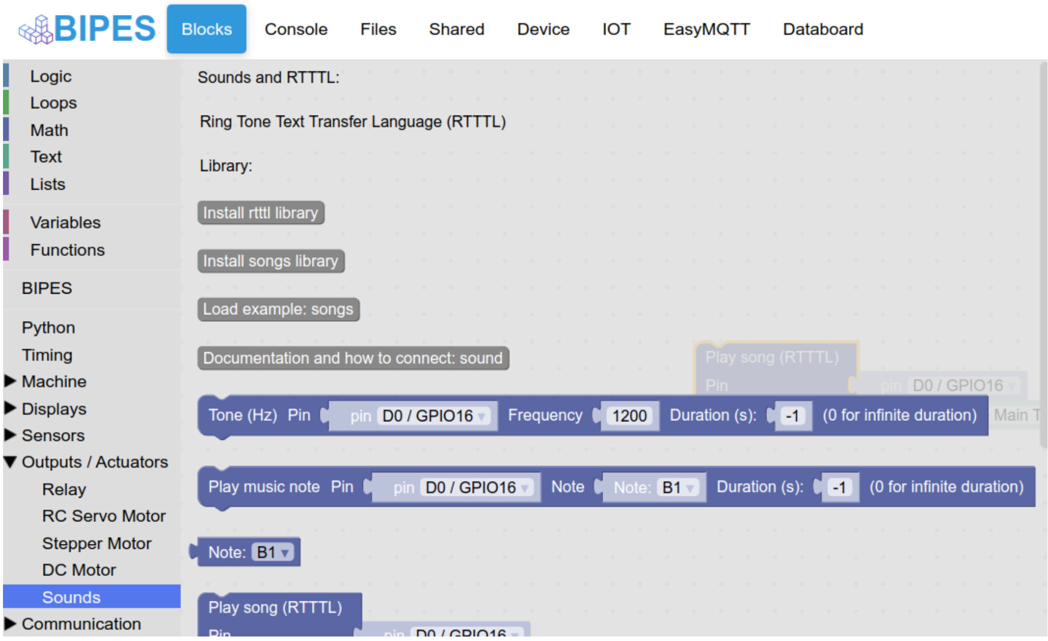


Challenge:
Implement a “gimbal” type system that adjusts the servo’s position as the angle measured by the MPU6050 inertial unit (in this case, using accelerometers only).

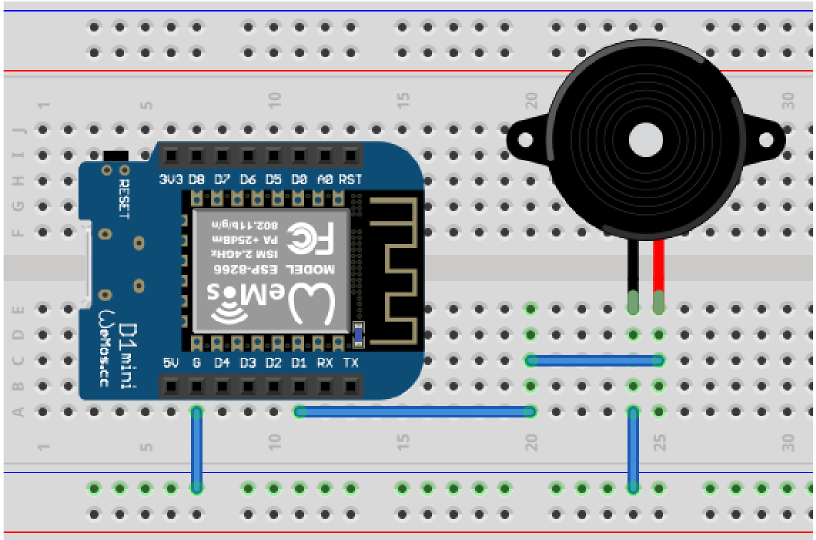
Music

BIPES also offers blocks to generate sound tones and play complete songs! Check blocks and options on the side toolbox **Outputs and Actuators >> Sounds**. About the hardware to play a song, just use a *buzzer*.

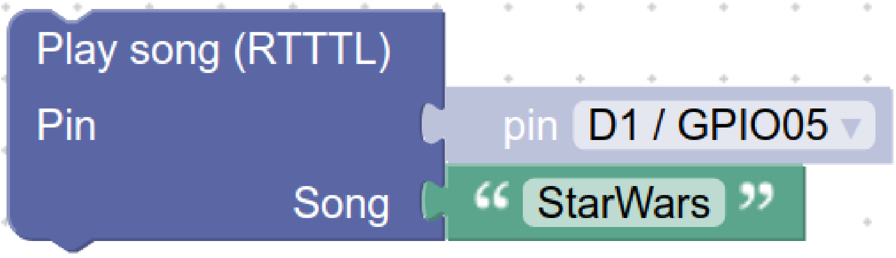
To produce sounds, **BIPES** uses an external library. Go to the side tab **Outputs and Actuators >> Sounds**, click **Install rtttl library** and then **Install songs library**. Two files will be downloaded and installed in the boards' memory for audio playback. The board must be connected to the Internet to execute these commands.



After that, you can assemble a circuit with the ESP8266 board and a *buzzer*, as suggested below:

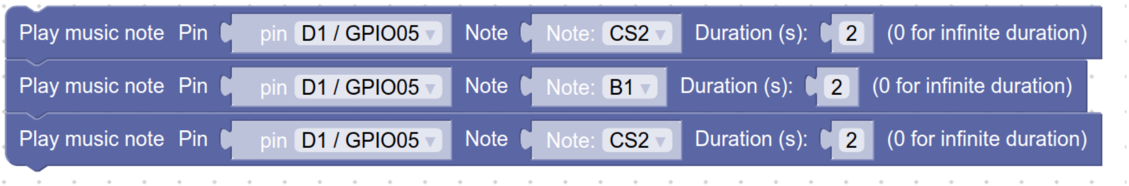


Then you can play songs! Note that there is a library with several ready-made songs, which you can use directly, or even include/create your songs by editing the songs.py file.



Play song (RTTTL)
Pin: pin D1 / GPIO05
Song: “ StarWars ”

It is also possible to play a sequence of musical notes playing your music:



Play music note Pin pin D1 / GPIO05 Note Note: CS2 Duration (s): 2 (0 for infinite duration)
Play music note Pin pin D1 / GPIO05 Note Note: B1 Duration (s): 2 (0 for infinite duration)
Play music note Pin pin D1 / GPIO05 Note Note: CS2 Duration (s): 2 (0 for infinite duration)

We have a video on YouTube with examples of some songs being played by an ESP32 card: https://www.youtube.com/watch?v=a7U-sz70Wrg

BIPES with other platforms

BIPES is available for several other platforms and offers several other possibilities, including closed-loop PID control blocks, RFID, computer vision with OpenCV, among others. So explore, enjoy, share and contribute to the project!

Python with Arduino - Snek

BIPES also works with Arduino, thanks to Snek Lang (https://sneklang.org/) by Keith Packard. Check more at: https://bipes.net.br/snek-web-uploader/.

Others

BIPES supports multiple microcontroller platforms and SoCs *(Systems on* Chip), as micro:bit, STM32, and others. **BIPES** can also be used with Raspberry Pi Pico boards with MicroPython or Raspberry Pi (and similar) with Linux. In addition, BIPES was the first project to allow block-based programming for the Raspberry Pi Pico. Check the BIPES project website for more details and support hardware platforms: https://bipes.net.br/.

Getting help and helping

The **BIPES** project has a user support community on GitHub. If you have any questions, criticisms, or suggestions, visit our community and feel free to ask, help, and contribute to the project through the link:

https://github.com/BIPES/BIPES/discussions/

You can also contribute to **BIPES** in several ways. Learn more about the project at https://github.com/BIPES and https://bipes.net.br/.

Final remarks

Embedded systems and Internet of Things (IoT) applications can be useful on many occasions. BIPES aims to facilitate rapid prototyping of embedded and IoT applications and contribute to more accessible and more intuitive access to programming these systems for less experienced users. In addition, BIPES is also expected to increase the productivity of experienced users.

Science, Technology, Engineering, Arts and Math (STEM or STEAM) projects can benefit from BIPES. For example, it can contribute to educational robotics projects and *maker* projects, offering intuitive programming, data collection, data storage, a visualization platform, and eliminating the need to install software or make specific configurations, facilitating its use in different types of computers and devices. It also offers a different way to learn programming by code since block-based programs are automatically converted to Python. In addition, the Python code can be edited, modified, and quickly tested.

In the academic environment, it can be used to automate laboratory experiments, collect data, or facilitate laboratory practices that need some monitoring or automation.

This book was also based on the use of low-cost devices, such as ESP8266 and ESP32 modules, already widely used in home automation, commercial, educational, academic, and, in some cases, machine monitoring applications in industries, in the field, and other environments.

Another pillar of the BIPES project is the MicroPython or its variant, the CircuitPython. Although it is a version of Python optimized for microcontrollers, there are several success stories of industrial applications using MicroPython and even space projects based on MicroPython. The European Space Agency (ESA), for example, has a project to use MicroPython in space applications[4]. There are also several educational and robotics kits using MicroPython. For instance, educational satellite kits based on ESP32 processors, such as educational satellites from PION, MySatKit, and IdeaSpace.

We hope this text was helpful, and we appreciate your attention!

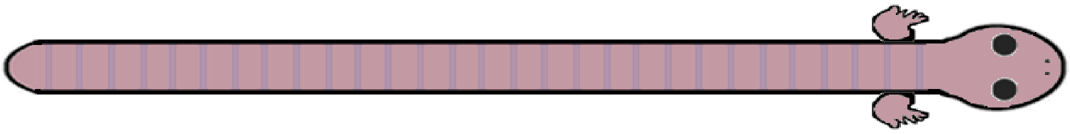
---

[4] https://essr.esa.int/project/micropython-for-leon-pre-qualified-version

Acknowledgments

The project **BIPES** integrates several open-source software tools: **Google Blockly**, **MicroPython**, **freeboard**, **CodeMirror**, MicroPython libraries, esp-web-tools, among others. The **BIPES** team leaves a big thank you to all the developers of these free tools. We thank the **BIPES** project team colleagues, including Gustavo Tamanaka, for the project logo and Caio Augusto Silva for implementing the EasyMQTT module. We are grateful to the project **Fritzing**, which made the drawings and diagrams in this text possible. We are thankful for the support of the Federal University of São Carlos through its Dean of Extension. We also thank the National Council for Scientific and Technological Development of the Ministry of Science, Technology, and Innovation (CNPq/MCTI) for supporting the **BIPES** project (CNPq grant 306315/2020-3).

BIPES



Block based Integrated Platform for Embedded Systems

http://bipes.net.br

Support:

CNPq
FUNDAÇÃO VINCULADA AO MCTI

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÕES

PATRIA AMADA
BRASIL
GOVERNO FEDERAL

An introduction to the Internet of Things and Embedded Systems using block programming with BIPES and ESP8266 / ESP32 (https://bipes.net.br)

December/2021 - 1st Edition

Rafael Vidal Aroca
Wesley Flavio Gueta
Jorge André Gastmaier Marques
Tatiana de Figueiredo Pereira Alves Taveira Pazelli

ISBN: 978-65-00-36922-9
9 786500 369229