

**UNIVERSIDADE DE SÃO PAULO
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e de Computação**

SEL0337/SEL0630

PROJETOS EM SISTEMAS EMBARCADOS

Capítulo 4

**Introdução à Programação em Sistemas
Embarcados**

GPIO, Sensores e Periféricos e Processos

Prof. Pedro de Oliveira C. Junior

pedro.oliveiracjr@usp.br

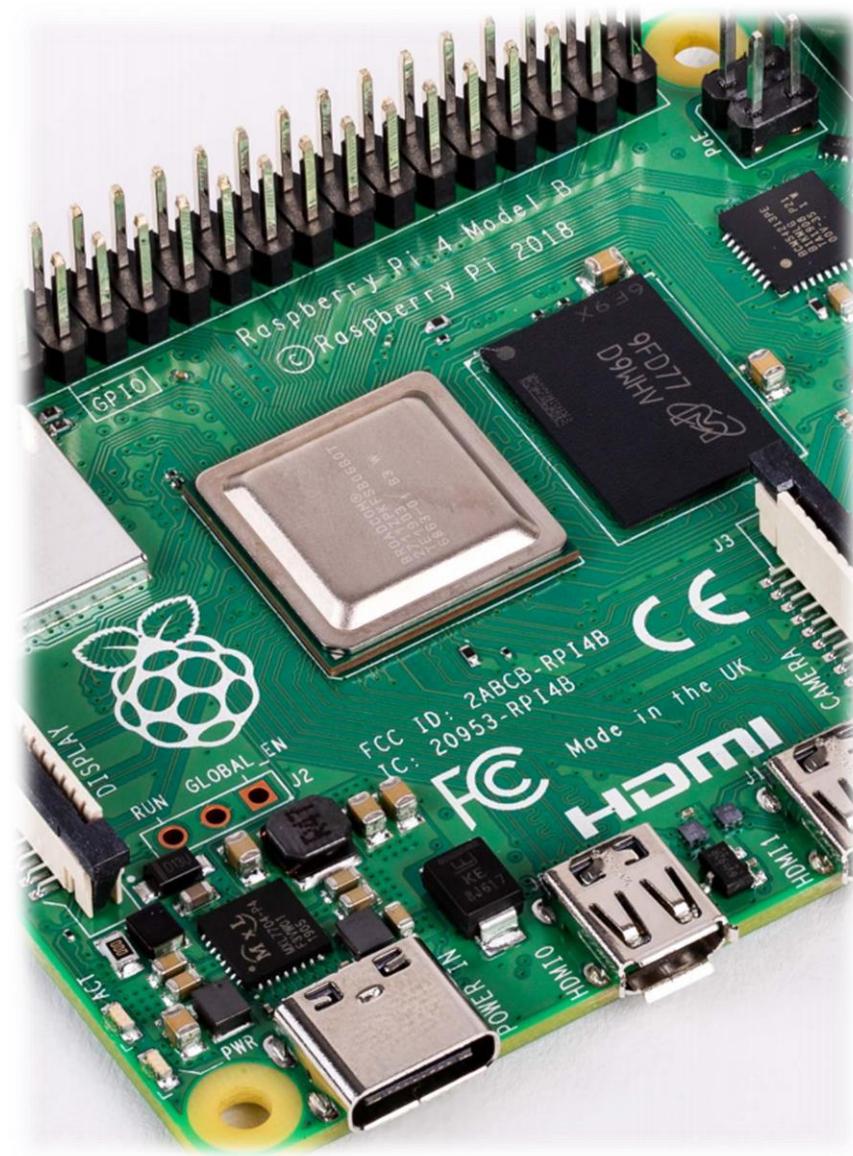
OBJETIVOS



Introdução à programação em Python
para sistemas embarcados;

Introdução ao uso da GPIO da Raspberry Pi: importação de bibliotecas, ambiente virtual, I/O digital, detecção de eventos, sensores para automação (temperatura, umidade, movimento, distância etc.)

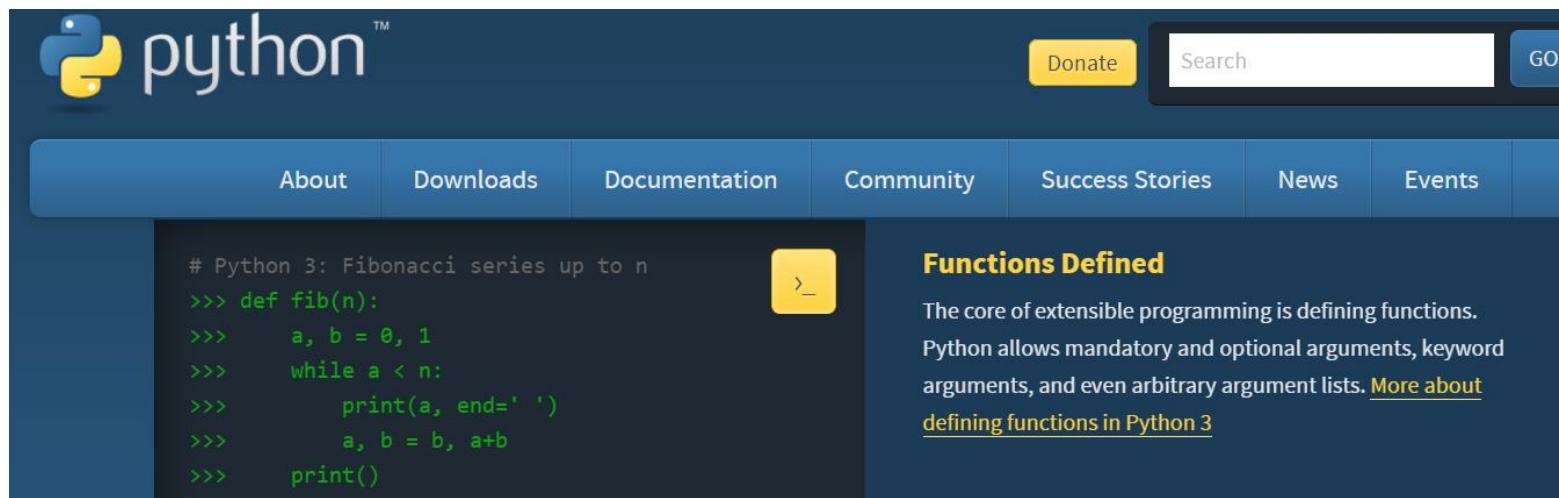
Computação paralela, processos e threads



Linguagem de Programação Python

- ✓ Linguagem de programação de **alto nível; interpretada; orientada a objetos, de tipagem dinâmica e forte!**
- ✓ Lançada em 1991 por Guido Van Rossum;
- ✓ Atualmente em **modelo open**, mantida pela org. sem fins lucrativos **Python Software Foundation.**

➤ Documentação, comunidade e downloads: <https://www.python.org>



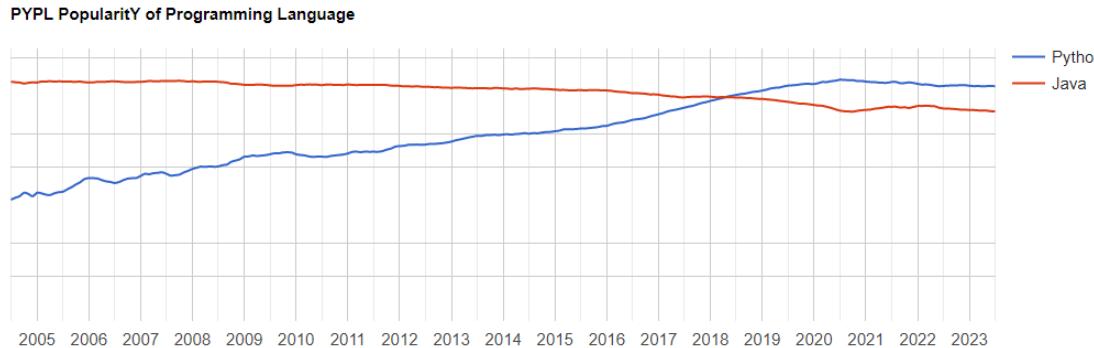
The screenshot shows the Python.org homepage with a dark blue header. The header features the Python logo, the word "python" in white, and navigation links for "About", "Downloads", "Documentation", "Community", "Success Stories", "News", and "Events". On the right side of the header are "Donate", "Search", and "GO" buttons. Below the header, there's a search bar with placeholder text "Search" and a magnifying glass icon. The main content area has a dark background with green code snippets. On the left, a code editor window displays Python code for generating a Fibonacci series up to n:

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

To the right of the code, a yellow button with a double-right arrow icon is positioned above a section titled "Functions Defined". This section contains text about defining functions in Python, mentioning mandatory and optional arguments, keyword arguments, and arbitrary argument lists, with a link to "More about defining functions in Python 3".

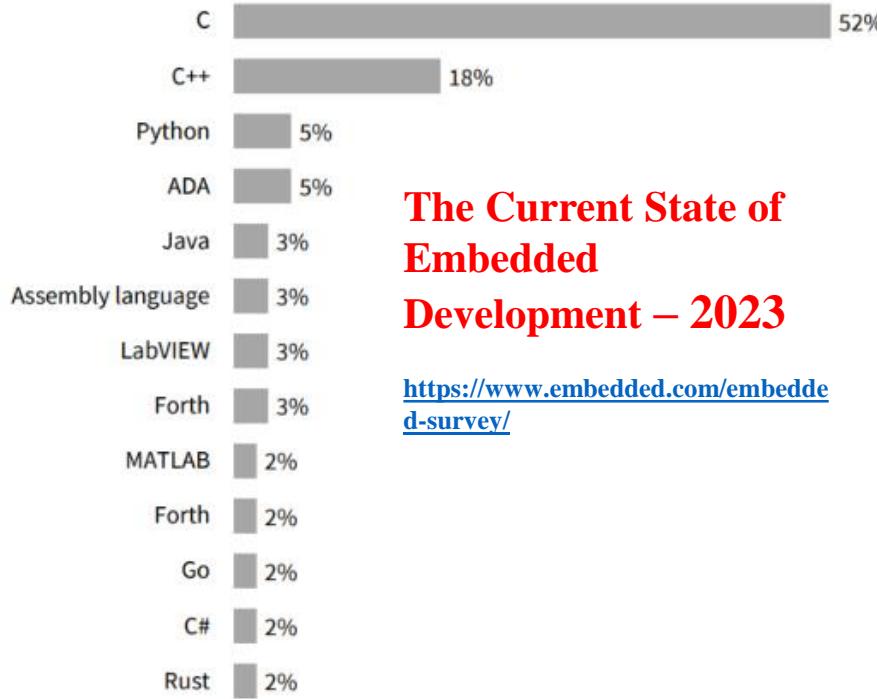
Popularidade da linguagem Python

- PYPL PopularitY of Programming Language (Worldwide)
- <https://pypl.github.io/PYPL.html>



Worldwide, Aug 2024 :		
Rank	Change	Language
1		Python
2		Java
3		JavaScript
4		C#
5		C/C++
6	↑	R
7	↓	PHP
8		TypeScript
9		Swift
10	↑	Rust
11	↓	Objective-C
12		Go
13		Kotlin
14		Matlab

Popularidade da linguagem Python



The Current State of Embedded Development – 2023

<https://www.embedded.com/embedded-survey/>



Relatório de Pesquisa sobre o Mercado Brasileiro de Sistemas Embarcados – 2023

<https://embarcados.com.br/relatorio-da-pesquisa-sobre-o-mercado-brasileiro-de-sistemas-embarcados-e-iot-2023/>

Linguagem “C” vs. Python

- ✓ Um compilador traduz linguagem Python em linguagem de máquina – o código Python é traduzido em um código intermediário que deve ser executado por uma máquina virtual conhecida como PVM (Python Virtual Machine) (similar ao Java)



PROGRAMAÇÃO ESTRUTURADA

HIGH SPEED

**MENOR QTDE DE FUNÇÕES DE
BIBLIOTECAS**

HARDER SYNTAX

LING. DE DOMÍNIO ESPECÍFICO



LING. INTERPRETADA

SLOW SPEED

RICH LIBRARY

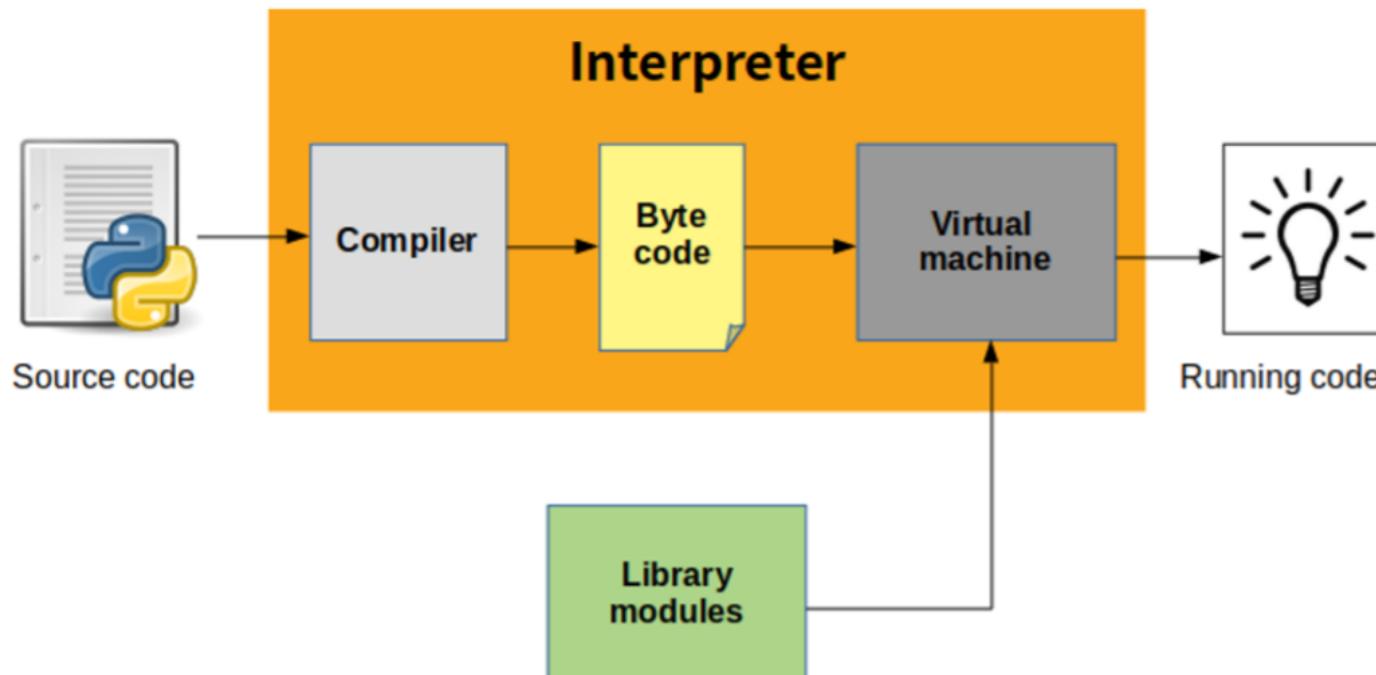
EASY SYNTAX

Vs.

LING. DE PROPÓSITOS DIVERSOS

Compilação/Interpretação em Python

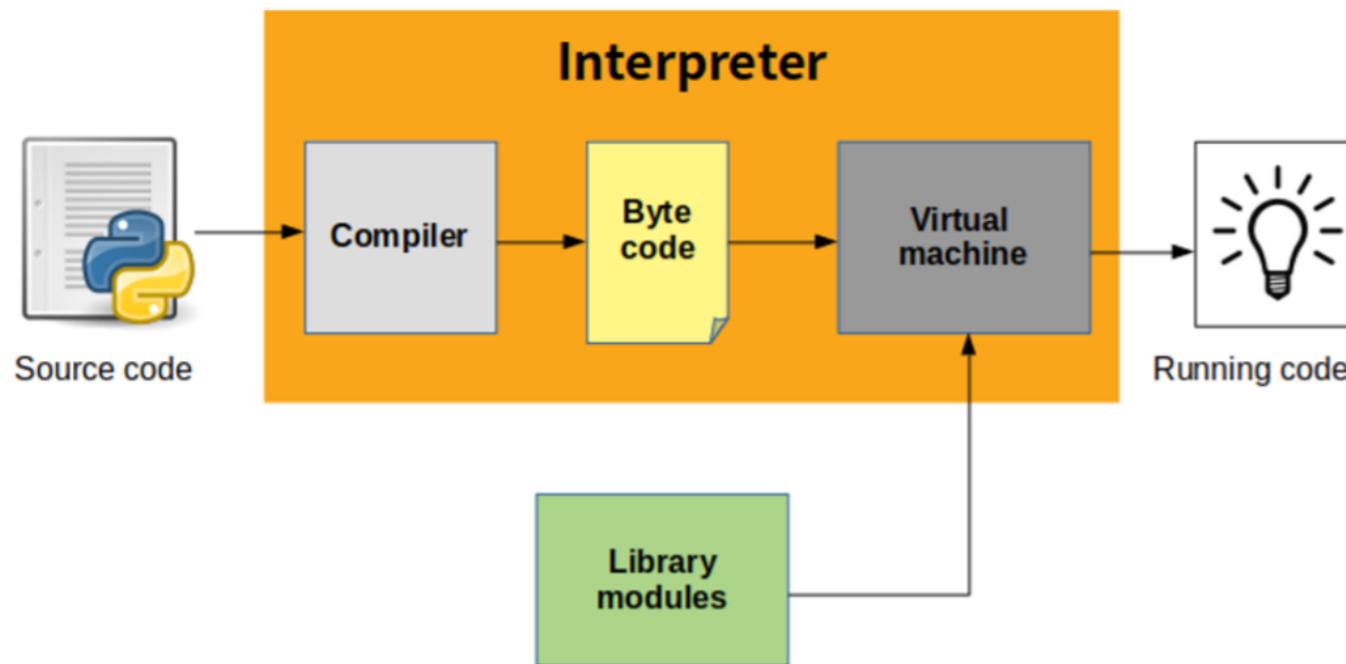
- ✓ 1 - Primeiramente o código “.py” é traduzido para bytecode, que é um formato multiplataforma binário com instruções para o interpretador.



<https://indianpythonista.wordpress.com/2018/01/04/how-python-runs/>

Compilação/Interpretação em Python

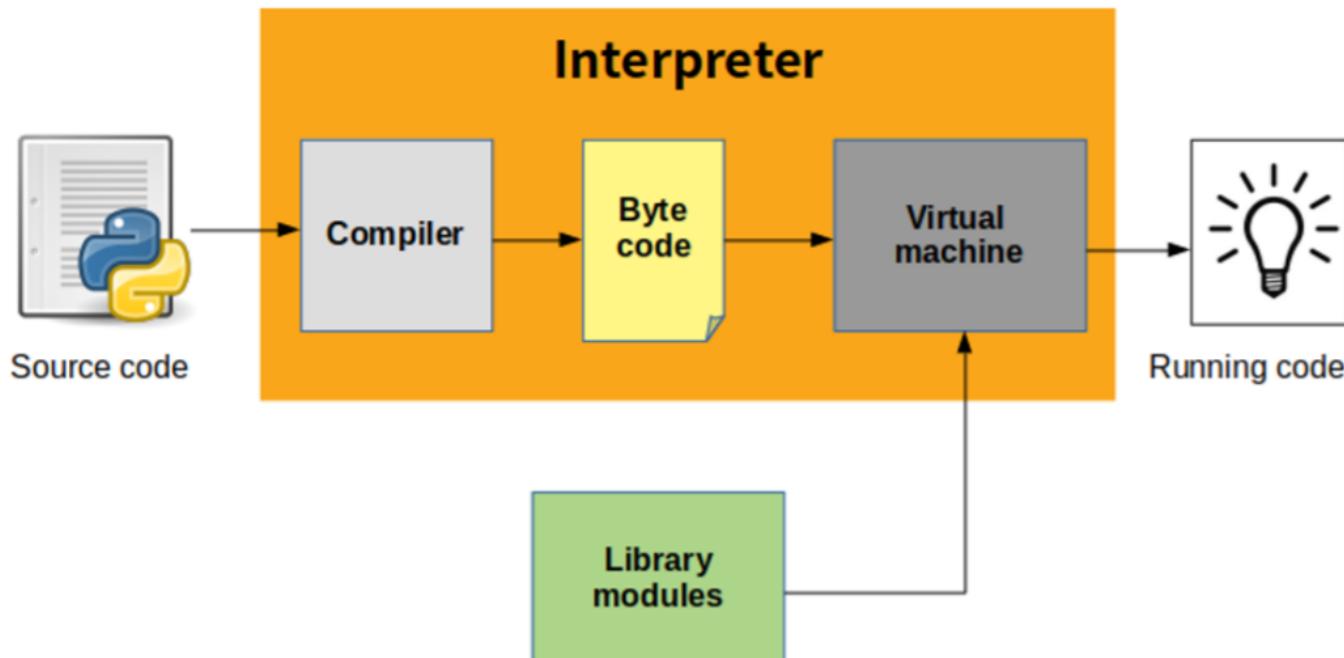
- ✓ 2 - o interpretador compila o código e armazena o bytecode em disco. Nesta etapa, o interpretador analisa o código, converte para os símbolos padrões, compila e executa na máquina virtual do Python (Virtual Machine).



<https://indianpythonista.wordpress.com/2018/01/04/how-python-runs/>

Compilação/Interpretação em Python

- ✓ 3 - o interpretador Python é denominado Cpython, responsável por realizar a implementação da linguagem Python por meio de um pacote de ferramentas



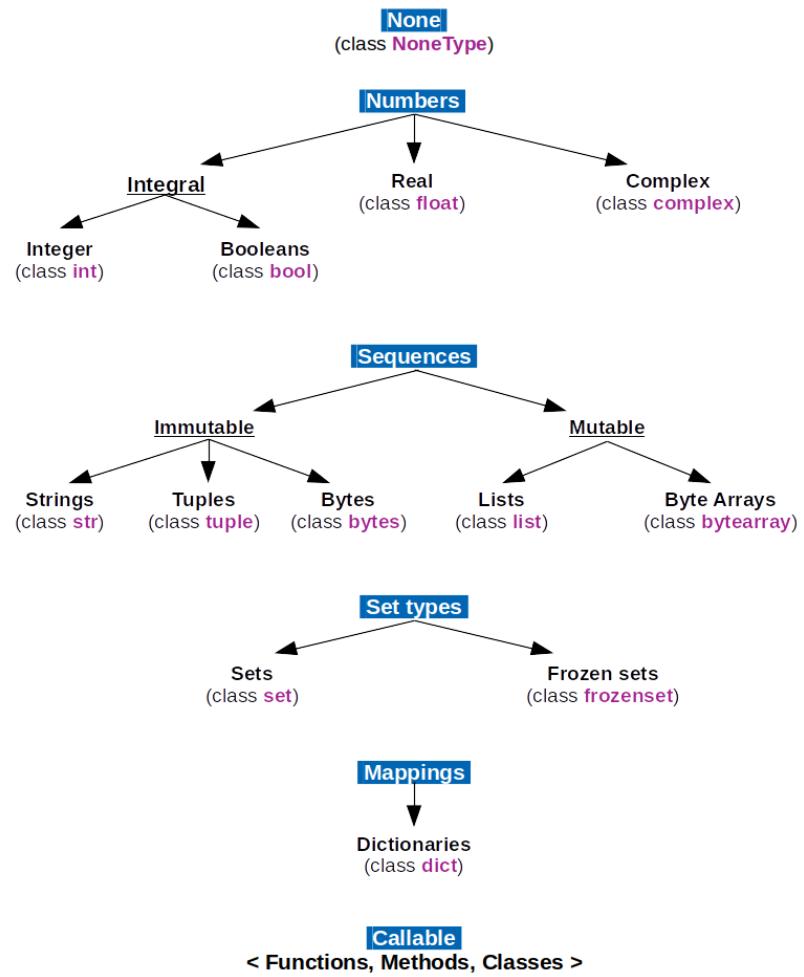
<https://indianpythonista.wordpress.com/2018/01/04/how-python-runs/>

Hierarquia e cultura Python



<https://www.geeksforgeeks.org/python2-vs-python3-syntax-and-performance-comparison/>

Python 3 The standard type hierarchy



https://upload.wikimedia.org/wikipedia/commons/1/10/Python_3._The_standard_type_hierarchy.png

Tipos de dados em Python

Tipo de dado	Descrição	Exemplo da sintaxe
<i>bool</i>	<u>Booleano</u>	<code>True</code> ou <code>False</code>
<i>int</i>	<u>Número de precisão fixa, é transparentemente convertido para <i>long</i> caso não caiba em um <i>int</i>.</u>	<code>42</code> <code>2147483648L</code>
<i>float</i>	<u>Ponto flutuante</u>	<code>3.1415927</code>
<i>complex</i>	<u>Número complexo</u>	<code>3+2j</code>
<i>list</i>	<u>Lista heterogênea mutável</u>	<code>[4.0, 'string', True]</code>
<i>range</i>	<u>Sequência de números imutável que pode ser transformada em lista</u>	<code>range(10)</code> <code>range(0, 10)</code> <code>range(0, 10, 1)</code>
<i>set, frozenset</i>	<u>Conjunto não ordenado, não contém elementos duplicados</u>	<code>{4.0, 'string', True}</code> <code>frozenset([4.0, 'string', True])</code>
<i>str, unicode</i>	<u>Uma cadeia de caracteres imutável</u>	<code>'SEL'</code> <code>u'SEL'</code>
<i>bytes, bytearray, memoryview</i>	<u>Sequência binária</u>	<code>b'SEL'</code> <code>bytearray(b'SEL')</code> <code>memoryview(b'SEL')</code>
<i>dict</i>	<u>Conjunto associativo</u>	<code>{'key1': 1.0, 'key2': False}</code>

Palavras reservadas e indentação, objetos

Palavras reservadas

```
False      None      True      and      as
assert     break     class     continue
def        del       elif      else     except
finally    for       from     global   if
import    in        is        lambda  not
nonlocal  or        pass     raise   try
return   while    with     yield
```

Exemplo de comentários

```
# comentários
# ...
# SEL0337 / SEL0630

import time
from gpiozero import LED

var = LED(18)

while True:
    var.on()
    time.sleep(1)
    var.off()
    time.sleep(1)
```

```
def valor1():
    while True:
        try:
            c = int(input('Primeiro
Valor: '))
            return c
        except ValueError:
            print 'Inválido!'
```

Indentação correta



```
try:
    while true:
        if:
            elif:
                else:
                    finally:
```

Operadores em Python - exemplos

- Exemplos de uso de operadores em Python:

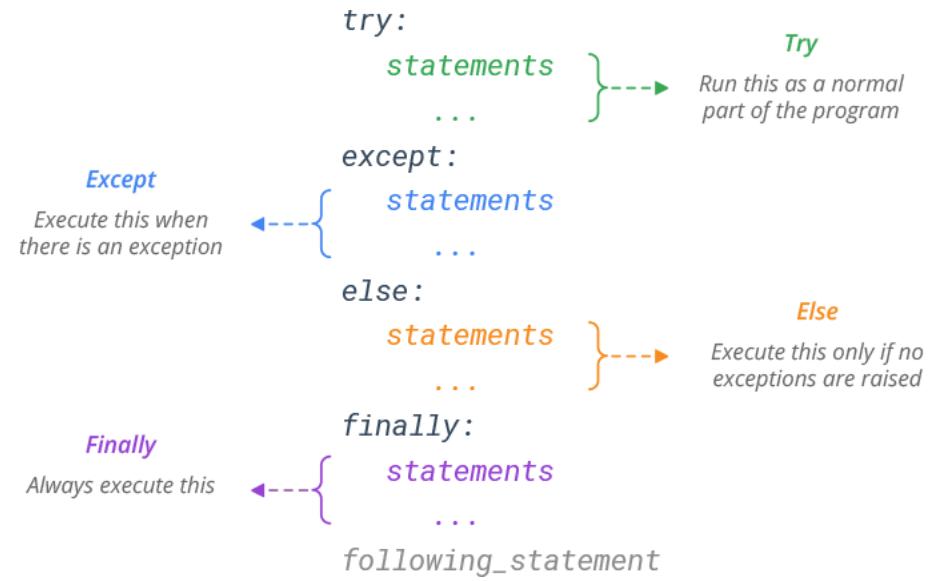
<https://datascienceparichay.com/article/python-operators/>

- Functions e cláusulas “Try Except”:

```
try:
    ham = spam.eggs
except AttributeError:
    handle_error()
```

```
def add(x, y):
    print(f'arguments are {x} and {y}')
    return x + y
```

1. def keyword
2. function name
3. function arguments inside ()
4. colon ends the function definition
5. function code
6. function return statement



<http://digital.academy.free.fr/blog/python-exceptions/>

Instalação e “type casting”

➤ **Instalação:**

```
sudo apt update  
sudo apt install python3
```

➤ **Interpretação de códigos:**

```
python helloworld.py
```

➤ **Verificando versão e ajuda**

```
python -version  
Python --help
```

➤ **Gerenciamento de pacotes com PIP**

```
pip install Django  
pip freeze >> requirements.txt
```

➤ **Especificação de formato (ex.: data)**

```
def dateformat(date):  
    day, month, year=date.split('/')  
    return "{:4d}{:02d}{:02d}".format(int(year),int(month),int(day))
```

Documentação e material de revisão

- Documentação: <https://docs.python.org/3/index.html>
- Standard Library <https://docs.python.org/3/library/index.html>
- Built-in Types: <https://docs.python.org/3/library/stdtypes.html>
- String operations: <https://docs.python.org/3/library/string.html>
- **Curso de Computação científica em Python (curso USP Lorena): <https://computeel.org/LOM3260/>**



The screenshot shows the homepage of a scientific computing course. The header reads "Computação Científica em Python". Below it, a sub-header states: "Uma disciplina do curso de Engenharia Física da Escola de Engenharia de Lorena (EEL) da Universidade de São Paulo (USP)". A sidebar on the left lists topics: "Sobre a disciplina", "Tema 1", "Tema 2", "Tema 3", "Tema 4", "Tema 5", "Tema 6", "Tema 7", and "Exercícios". The main content area features a paragraph about the site's purpose and a note about its development status. At the bottom is a photograph of a brick building with a sign that says "DEMAR INSTITUTO PRINCIPAL".

IDEs

- Top 10 IDE index: <https://pypl.github.io/IDE.html>
- Guia: <https://realpython.com/python-ides-code-editors-guide/>



➤ Sublime Text

<https://www.sublimetext.com>



<https://www.spyder-ide.org>



Visual Studio Code

<https://code.visualstudio.com>



<https://jupyter.org>



PyCharm

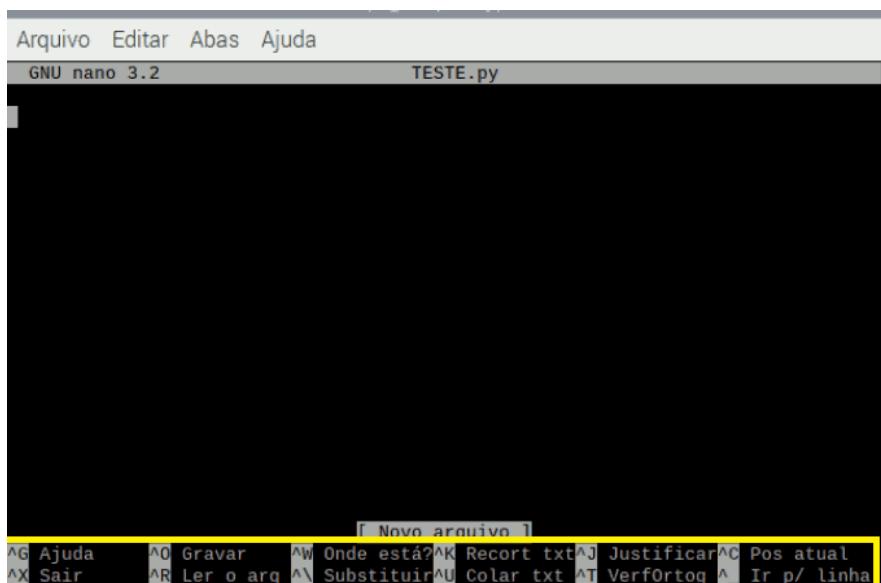
<https://www.jetbrains.com/pt-br/pycharm/download/#section=windows>

Worldwide, Sept 2023 :

Rank	Change	IDE
1		Visual Studio
2		Visual Studio Code
3		Eclipse
4	↑	pyCharm
5	↓	Android Studio
6		IntelliJ
7		NetBeans
8	↑	Xcode
9	↓	Sublime Text
10		Atom

Uso do editor “nano” e outras IDEs

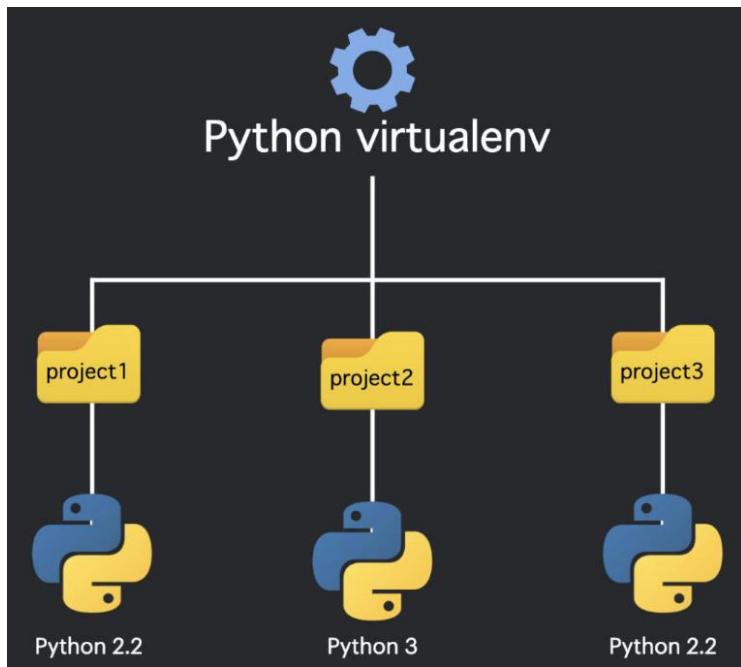
- O uso de uma IDE da lista anterior automatiza o trabalho e possibilita muitos recursos de alto nível, como identação, sugestões de uso de comandos, integração com controle versão (uso do VS Code, com Git integrado, por exemplo) etc. Portanto, fica a critério do(a) aluno(a) o uso de uma dessas IDEs. Usaremos o nano ou a Thony IDE já instalada na Raspberry Pi em razão das limitações da placa “3B+” com IDEs mais robustas como as da lista anterior.



- "Ctrl + G" - Ao pressionar essas teclas, será aberto um documento de "Ajuda";
- "Ctrl + X" - Sair;
- "Ctrl + O" - Salva o arquivo que está sendo editado;
- "Ctrl + R" - Abre um outro arquivo pelo editor;
- "Ctrl + W" - permite que você encontre uma palavra específica;
- "Ctrl + \\" - permite substituir a palavra buscada por outra;
- "Ctrl + K" - Equivale ao comando "Ctrl + X" e recorta
- "Ctrl + U" - Equivale ao "Ctrl + V", e "cola" no arquivo
- "Ctrl + J" - o parágrafo em que o ponteiro está será justificado;
- "Ctrl + T" - verificador ortográfico, se estiver disponível
- "Ctrl + C" - Exibe a posição atual do ponteiro do editor
- "Ctrl + _" Posiciona o ponteiro do editor no começo da linha selecionada;

Criação de um ambiente virtual em Python

- Um V.E. em Python possibilita o isolamento de pacotes, evita conflitos de dependências e manter o ambiente global limpo.
- <https://virtualenv.pypa.io/en/latest/index.html>
- Criar um V.E. em python



```
sudo pip install virtualenv # instalação  
python3 -m venv my_VE # criação de  
um V.E. com nome “my_VE”  
#pode ser qualquer nome que preferir  
# ativação do VE:  
source my_VE/bin/activate  
#usar a linha de commandos acima  
sempre quando entrar no V.E.  
(my_VE) pip freeze # verifica libs  
(my_VE) cd my_VE  
(my_VE) ls  
(my_VE) deactivate # sai do ambiente
```

<https://blog.debugeverything.com/wp-content/uploads/2021/04/python-virtualenv-estrutura-projeto.jpg>

Conhecendo a GPIO da Raspberry Pi

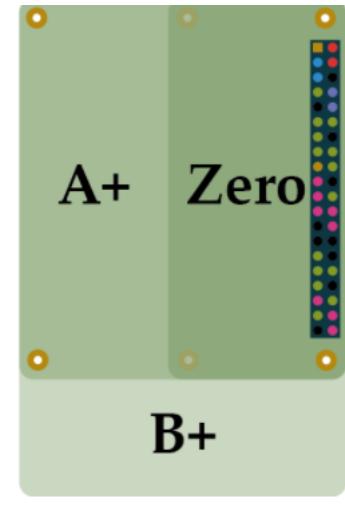
- <https://pinout.xyz>
- >> pinout

40 pinos entre I/O digital, saída 3V3 e 5V, GND, comunicação serial (UART, I2C, SPI), PWM etc.

Legend

Orientate your Pi with the GPIO on the right and the HDMI port(s) on the left.

- GPIO (General Purpose IO)
- SPI (Serial Peripheral Interface)
- I²C (Inter-integrated Circuit)
- UART (Universal Asynchronous Receiver/Transmitter)
- PCM (Pulse Code Modulation)
- Ground
- 5V (Power)
- 3.3V (Power)



3v3 Power	1	2	5v Power
GPIO 2 (I ² C1 SDA)	3	4	5v Power
GPIO 3 (I ² C1 SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)
Ground	9	10	GPIO 15 (UART RX)
GPIO 17	11	12	GPIO 18 (PCM CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3v3 Power	17	18	GPIO 24
GPIO 10 (SPI0 MOSI)	19	20	Ground
GPIO 9 (SPI0 MISO)	21	22	GPIO 25
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)
Ground	25	26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27	28	GPIO 1 (EEPROM SCL)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM DIN)
Ground	39	40	GPIO 21 (PCM DOUT)

GPIO – documentação

- <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#gpio-and-the-40-pin-header>

GPIO and the 40-pin Header

Voltages

Outputs

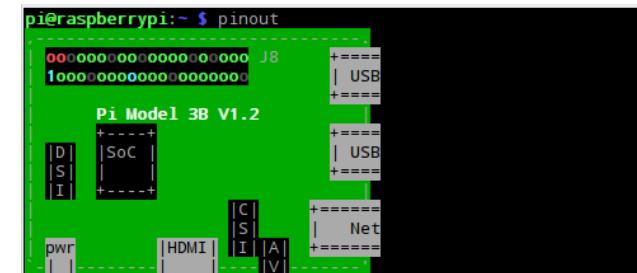
Inputs

More

GPIO pinout

Permissions

GPIO in Python



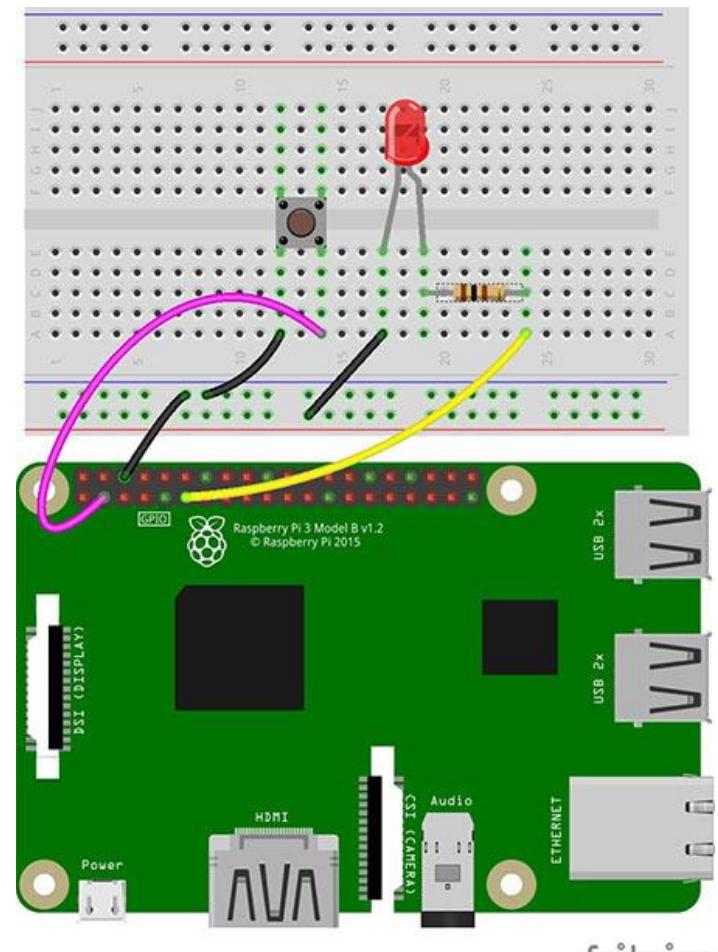
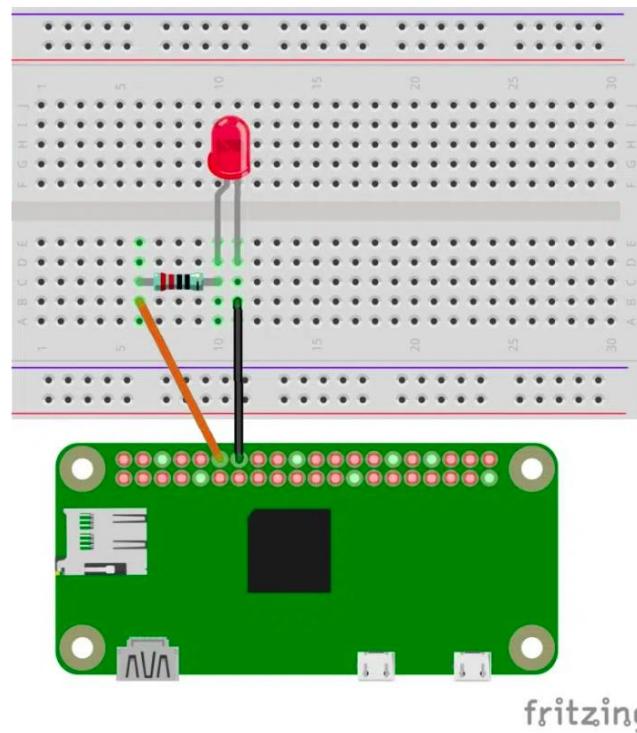
```
pi@raspberrypi:~ $ pinout
 00000000000000000000 J8
10000000000000000000
Pi Model B V1.2
+---+
|D| |SoC|
|S| |
|I| +---+
pwr |HDMI| |C| |Net
    |     |S| |A| +----+
    |     |V| +----+
Revision : a02082
SoC      : BCM2837
RAM      : 1024Mb
Storage   : MicroSD
USB ports : 4 (excluding power)
Ethernet ports : 1
Wi-fi     : True
Bluetooth : True
Camera ports (CSI) : 1
Display ports (DSI): 1

J8:
 3V3  (1) (2) 5V
GPIO02 (3) (4) 5V
GPIO03 (5) (6) GND
GPIO04 (7) (8) GPIO14
GND   (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
 3V3  (17) (18) GPIO24
GPIO10 (19) (20) GND
GPIO09 (21) (22) GPIO25
GPIO11 (23) (24) GPIO08
GND   (25) (26) GPIO07
GPIO00 (27) (28) GPIO01
GPIO05 (29) (30) GND
GPIO06 (31) (32) GPIO012
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO016
GPIO026 (37) (38) GPIO020
GND   (39) (40) GPIO021

For further information, please refer to https://pinout.xyz/
pi@raspberrypi:~ $
```

Explorando o Fritzing

- <https://fritzing.org>
- >> sudo install fritzing



Manipulando a GPIO pelo sistema operacional

➤ Execute a seguinte atualização para compatibilidade dos exemplos abaixo:

```
>> sudo rpi-update cac01bed1224743104cb2a4103605f269f207b1a #6.1.54
```

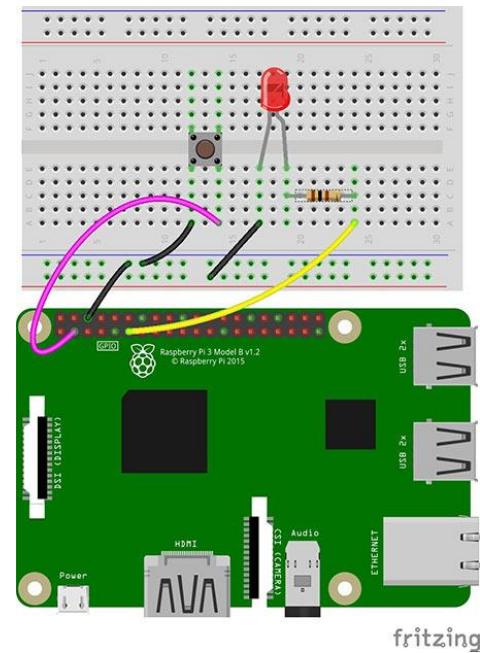
➤ Escrevendo na saída (GPIO 18):

- ✓ `echo 18 > /sys/class/gpio/export` Exporta o pino usado;
- ✓ `echo out > /sys/class/gpio/gpio18/direction` Define como saída
- ✓ `echo 1 > /sys/class/gpio/gpio18/value` Escreve o valor 1
- ✓ `echo 0 > /sys/class/gpio/gpio18/value` Escreve 0
- ✓ `echo 18 > /sys/class/gpio/unexport` Libera o pino

Exemplos usando diretamente o terminal Linux (comandos em bash)

➤ Lendo uma entrada (GPIO 18):

- ✓ `echo 18 > /sys/class/gpio/export` Exporta o pino usado;
- ✓ `echo in > /sys/class/gpio/gpio18/direction` Define como entrada
- ✓ `cat /sys/class/gpio/gpio18/value` Leitura do valor (0 ou 1)
- ✓ `echo 18 > /sys/class/gpio/unexport` Libera o pino

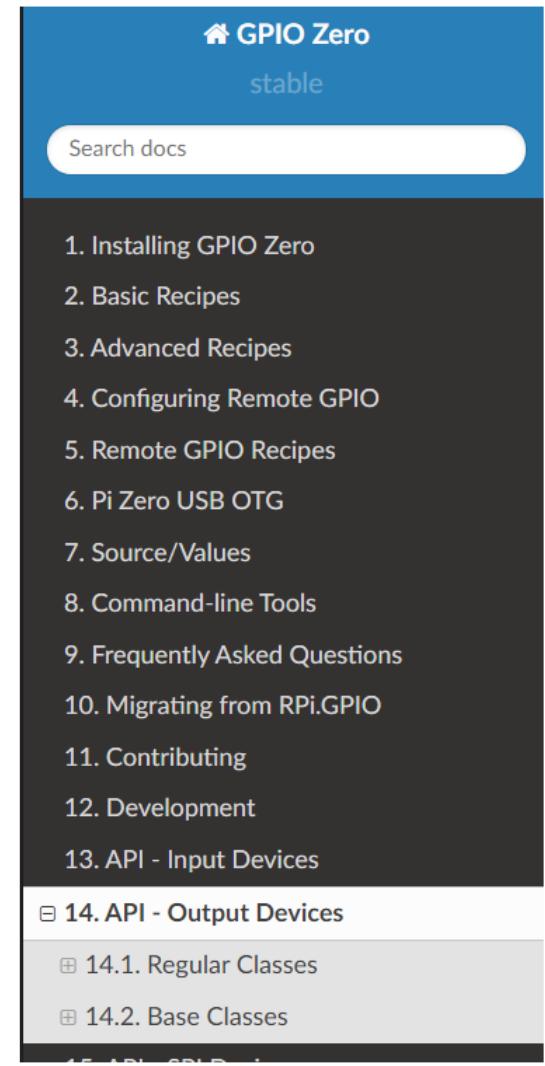


Bibliotecas para programação da GPIO em Python

➤ Python GPIO Zero

- <https://gpiozero.readthedocs.io/en/stable/index.html>
- pip install gpiozero
- pip freeze

led = LED(# of pin)	Determine the pin that is connected to the LED
on()	Turning on
off()	Turning off



The screenshot shows the official documentation for the `GPIO Zero` library. The top navigation bar includes a home icon, the text "GPIO Zero", and the word "stable". A search bar labeled "Search docs" is also present. The main content area displays a numbered table of contents:

1. Installing GPIO Zero
2. Basic Recipes
3. Advanced Recipes
4. Configuring Remote GPIO
5. Remote GPIO Recipes
6. Pi Zero USB OTG
7. Source/Values
8. Command-line Tools
9. Frequently Asked Questions
10. Migrating from RPi.GPIO
11. Contributing
12. Development
13. API - Input Devices
14. API - Output Devices
- 14.1. Regular Classes
- 14.2. Base Classes

Python gpiozero

➤ Introduz objetos abstratos:

LED, Button, Servo, Buzzer, MotionSensor...

```
from gpiozero import LED  
  
led = LED(17) # Cria um objeto LED no pino GPIO 17  
  
led.on()       # Liga o LED  
  
led.off()      # Desliga o LED
```

```
from gpiozero import Servo  
  
servo = Servo(17) # Cria um objeto Servo no pino GPIO 17  
  
servo.value = 0    # Define a posição do servo (0 a 1)
```

```
from gpiozero import MotionSensor, LED  
from signal import pause  
  
pir = MotionSensor(4)  
led = LED(16)  
  
pir.when_motion = led.on  
pir.when_no_motion = led.off  
  
pause()
```

1. Installing GPIO Zero

2. Basic Recipes

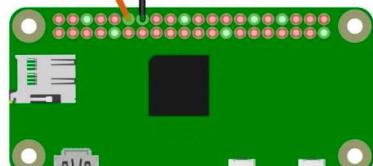
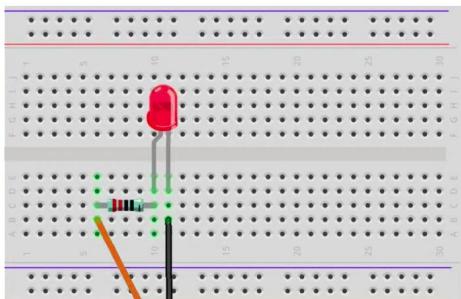
- 2.1. Importing GPIO Zero
- 2.2. Pin Numbering
- 2.3. LED
- 2.4. LED with variable brightness
- 2.5. Button
- 2.6. Button controlled LED
- 2.7. Button controlled camera
- 2.8. Shutdown button
- 2.9. LEDBoard
- 2.10. LEDBarGraph
- 2.11. LEDCharDisplay
- 2.12. Traffic Lights
- 2.13. Push button stop motion
- 2.14. Reaction Game
- 2.15. GPIO Music Box
- 2.16. All on when pressed
- 2.17. Full color LED
- 2.18. Motion sensor
- 2.19. Light sensor
- 2.20. Distance sensor

Python GPIO Zero

➤ Blink LED

➤ Itens necessários:

- ✓ 1x Protoboard; 1x LED;
- ✓ 1x resistor de 220Ω ;
- ✓ 1 x SBC Raspberry Pi (qualquer modelo).

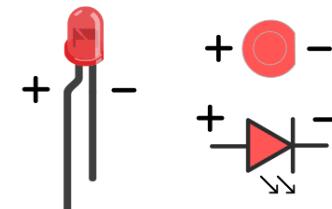


```
from gpiozero import LED  
from time import sleep
```

```
PinLED = LED(18)
```

```
while True:
```

```
    PinLED.on()  
    sleep(1)  
    PinLED.off()  
    sleep(1)
```



Python RPi.GPIO

- Biblioteca “**RPi.GPIO**”: fornece acesso mais direto à GPIO e menor abstração (configurações mais “manuais” e mais linhas de código e melhor compressão do hardware)
- Vide: https://gpiozero.readthedocs.io/en/stable/migrating_from_rpigpio.html
- pip3 install RPi.GPIO

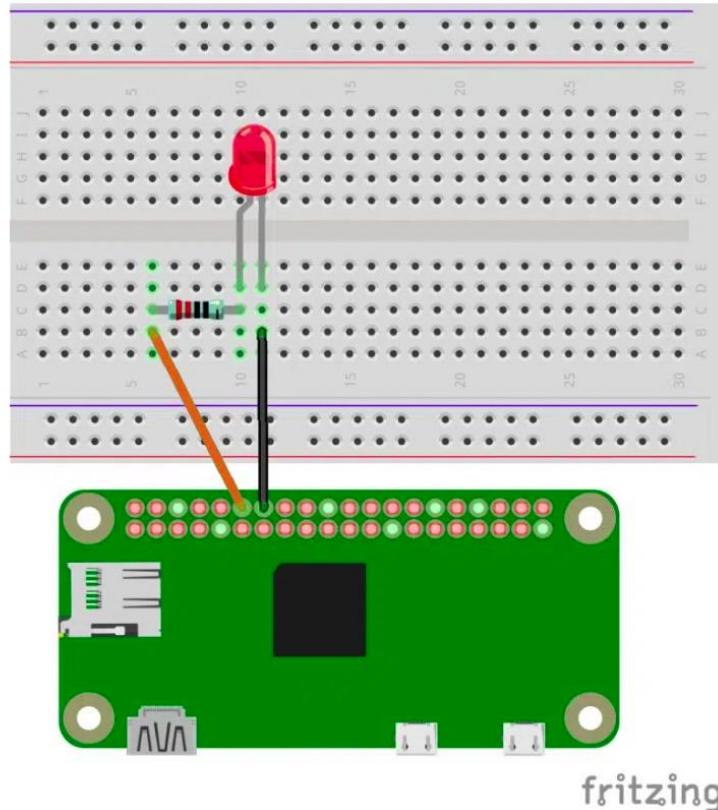
```
GPIO.setmode(GPIO.BCM)
```

- `GPIO.setmode(GPIO.BCM)` : Using pins BCM numbers
- `GPIO.setup(# of pin, GPIO.IN)` : Determine the pin as input
- `GPIO.setup(# of pin, GPIO.OUT)` : Determine the pin as an output
- `GPIO.setup(# of pin, GPIO.OUT, initial=GPIO.HIGH)` : Initialization output
- `GPIO.input(# of pin)` : Reading input pin
- `GPIO.output(# of pin, state)` : Writing on the output pin

<https://electropeak.com/learn/tutorial-raspberry-pi-gpio-programming-using-python-full-guide/>

Python RPi.GPIO

➤ Blink LED



```
import RPi.GPIO as GPIO #Define biblioteca da  
GPIO  
from time import sleep #Define biblioteca de  
tempo
```

```
GPIO.setmode(GPIO.BCM) # Define os pinos  
GPIO pelo canal do SoC da Broadcom
```

```
GPIO.setup(18, GPIO.OUT) #Define o pino 18  
da como saída
```

```
while True: #Inicia loop  
    GPIO.output(18, True) #Acende o LED  
    sleep(1) #aguarda 1 segundo  
    GPIO.output(18, False) #Apaga do LED  
    sleep(1) #Aguarda 1 segundo e reinicia o  
    processo
```

Python RPi.GPIO – Pino vs. n°GPIO

➤ Biblioteca “RPi.GPIO”

- ✓ **GPIO.BCM = N° GPIO**
- ✓ **GPIO.BOARD = N° DO PINO**
- ✓ **EXEMPLO: GPIO 24 (PINO 18)**

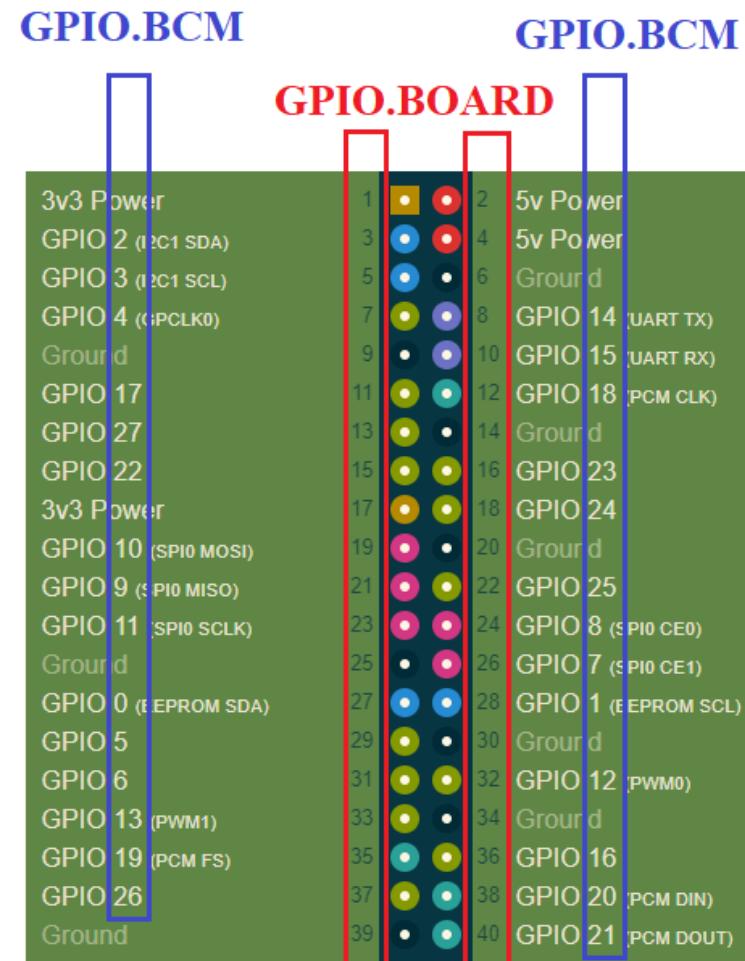
```
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(24, GPIO.IN)
```

```
#Ou
```

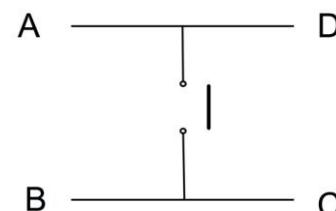
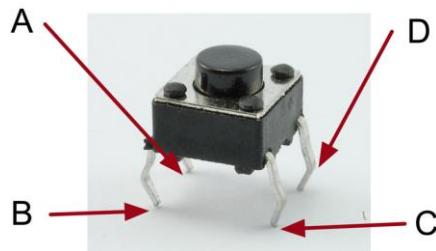
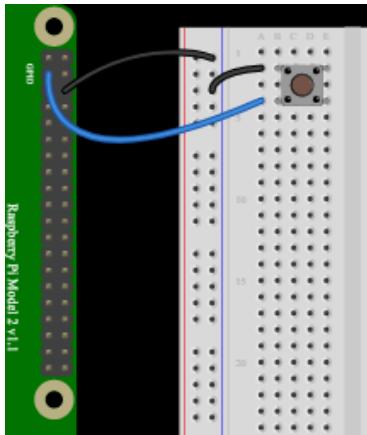
```
GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setup(18, GPIO.IN)
```



Detecção de eventos

- Botão usando “RPi.GPIO”



Fonte: <https://gpiozero.readthedocs.io/en/stable/recipes.html>

```

import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False) # desabilita avisos

def button(pin):
    if GPIO.input(pin):
        print ("released")
    else:
        # SE o botão foi pressionado:
        print("pressed")

GPIO.setup(4, GPIO.IN, GPIO.PUD_UP)
# GPIO.PUD_UP = "pull-up" / GPIO.PUD_DOWN para "pull-down"
GPIO.add_event_detect(4, GPIO.BOTH,
callback =button, bouncetime=200)
try:
    while True:
        pass
except KeyboardInterrupt:
    GPIO.cleanup()

```

Python RPi.GPIO - Detecção de eventos

➤ Tratamento de eventos: acionamento de um botão

```
#...
GPIO.add_event_detect(BT_PIN, GPIO.FALLING,
                      callback=button, bouncetime=50)
```

- ✓ Após inicializar a GPIO definida para o botão (entrada), a função “`add_event_detect`” registra o “`callback`” (retorno da chamada) para a interrupção nesta GPIO.

Parâmetros da função:

- ✓ **Canal:** número GPIO (modo BCM)
- ✓ **Tipo de interrupção:** “`GPIO.FALLING`”, “`GPIO.RISING`”, ou “`GPIO.BOTH`”.
- ✓ **Callback** (opcional): chamada quando uma interrupção é acionada (em outras situações o retorno pode ser chamado por outra função “`add_event_callback`”).
- ✓ **Bouncetime**, em milissegundos (opcional): no caso de várias ações serem acionadas em um curto período de tempo - devido ao retorno do botão - o retorno de chamada será chamado apenas uma vez. Ignora flutuações e trata efeito bounce em botões.

```
GPIO.add_event_detect(4, GPIO.BOTH, callback =button, bouncetime=200)
```

Python RPi.GPIO - Detecção de eventos

- Função **GPIO.add_event_detect**:

✓ Edge:

- GPIO.RISING (borda de subida)
- GPIO.FALLING(borda de descida)
- GPIO.BOTH (detecta ambas bordas)



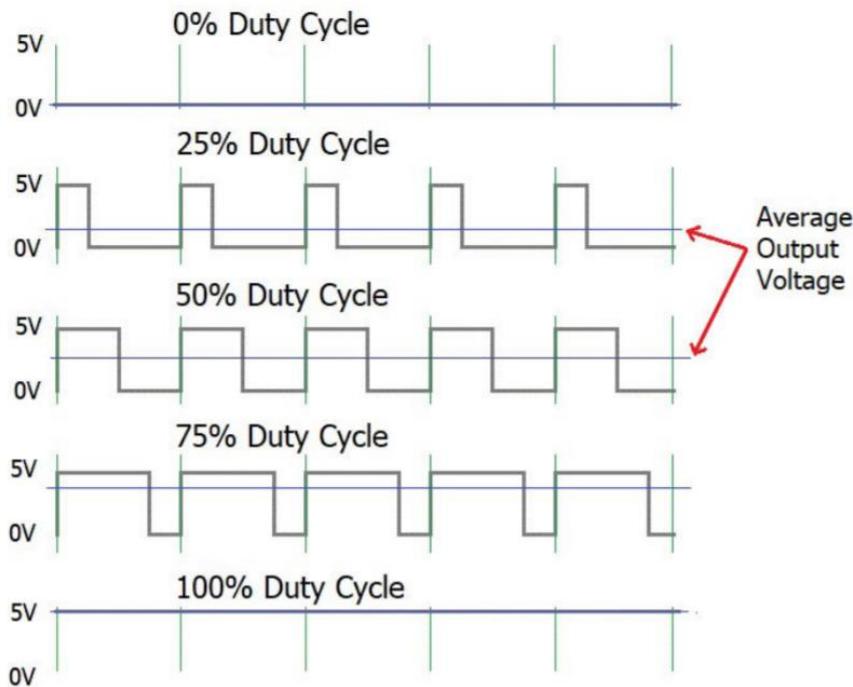
Python RPi.GPIO - Detecção de eventos

➤ Tratamento quando a execução do código é interrompida:

- ✓ “pass” é uma instrução vazia em Python, isto é, nenhuma operação é realizada, usada quando se deseja que o programa seja executado indefinidamente até ser interrompido ou como uma forma de manter o programa em execução enquanto aguarda um interrupção ou evento.
- ✓ KeyboardInterrupt é uma exceção que é disparada quando o usuário interrompe a execução de um programa pressionando a combinação de teclas Ctrl+C no teclado.
- ✓ GPIO.cleanup() # função da biblioteca RPi.GPIO que Limpa os pinos GPIO antes de encerrar o programa (importante para evitar resíduos de energia e inferência em outros programas que usar o mesmo pino, fazendo com que ele seja redefinido para o estado controlado da GPIO)

```
try:  
    while True:  
        pass  
except KeyboardInterrupt:  
    GPIO.cleanup()
```

PWM (*pulse width modulation*)

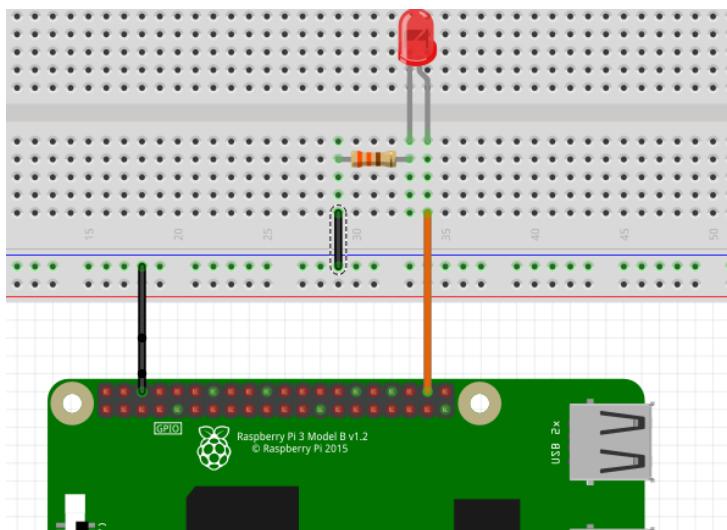


$$\text{DutyCycle} = 100 \cdot \frac{\text{LarguraPulso}}{\text{Periodo}}$$

$$\text{Periodo} = \text{LarguraPulso} + \text{TimeOFF}$$

Programação da GPIO usando Python

➤ PWM usando GPIO Zero



Fonte: <https://gpiozero.readthedocs.io/en/stable/recipes.html>

```
from gpiozero import PWMLED
from time import sleep

led = PWMLED(17)

while True:
    led.value = 0                      # off
    sleep(1)
    led.value = 0.5 # half brightness
    sleep(1)
    led.value = 1  # full brightness
    sleep(1)
```

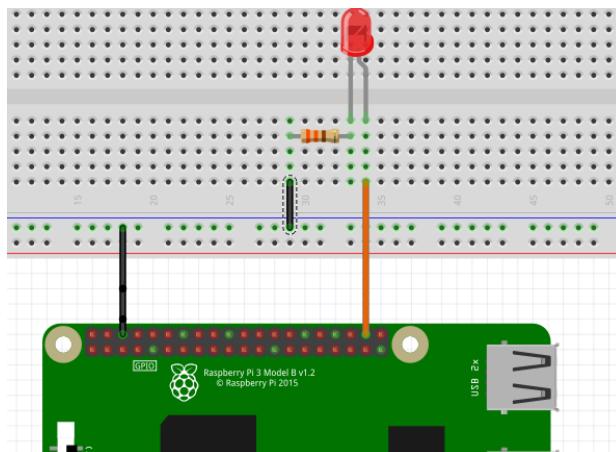
```
from gpiozero import PWMLED
from time import sleep
```

```
led = PWMLED(2)

for b in range(101):
    led.value = b / 100.0
    sleep(0.01)
```

Programação da GPIO usando Python

➤ PWM usando RPi.GPIO



GPIO.PWM (Pino, Frequência) → Define um Pino e uma frequência em Hertz para utilizar PWM.

NomeVar.start (Ciclo de trabalho [0-100]) → Define um ciclo de trabalho de início para variável escolhida.

NomeVar.ChangeDutyCycle (Ciclo de trabalho [0-100]) → Muda o ciclo de trabalho para a variável escolhida.

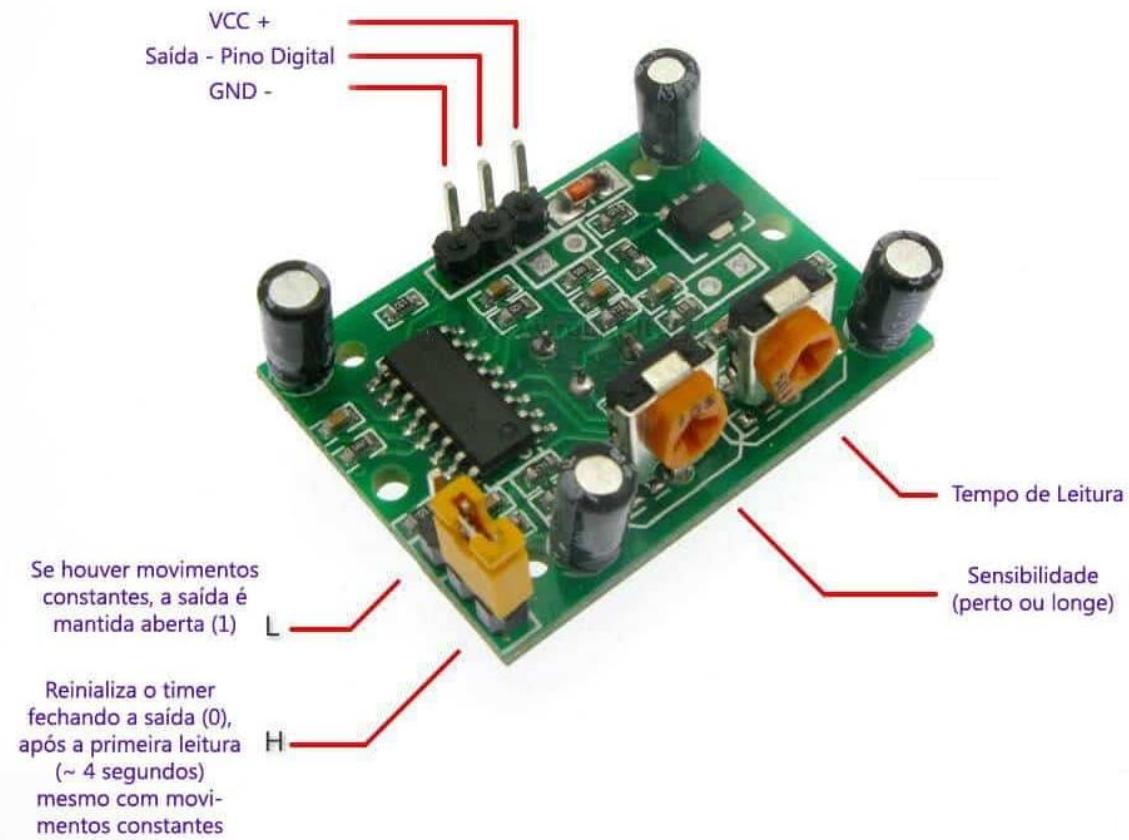
```
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(led_pin , GPIO.OUT)
```

```
pwm = GPIO.PWM(led_pin , 100)  
pwm.start (0)
```

Sensor de presença infravermelho (motion sensor) HC-SR501

➤ <https://www.epitran.it/ebayDrive/datasheet/44.pdf>



Sensor de presença (motion sensor) HC-SR501

```

from gpiozero import MotionSensor, LED
#importa as classes

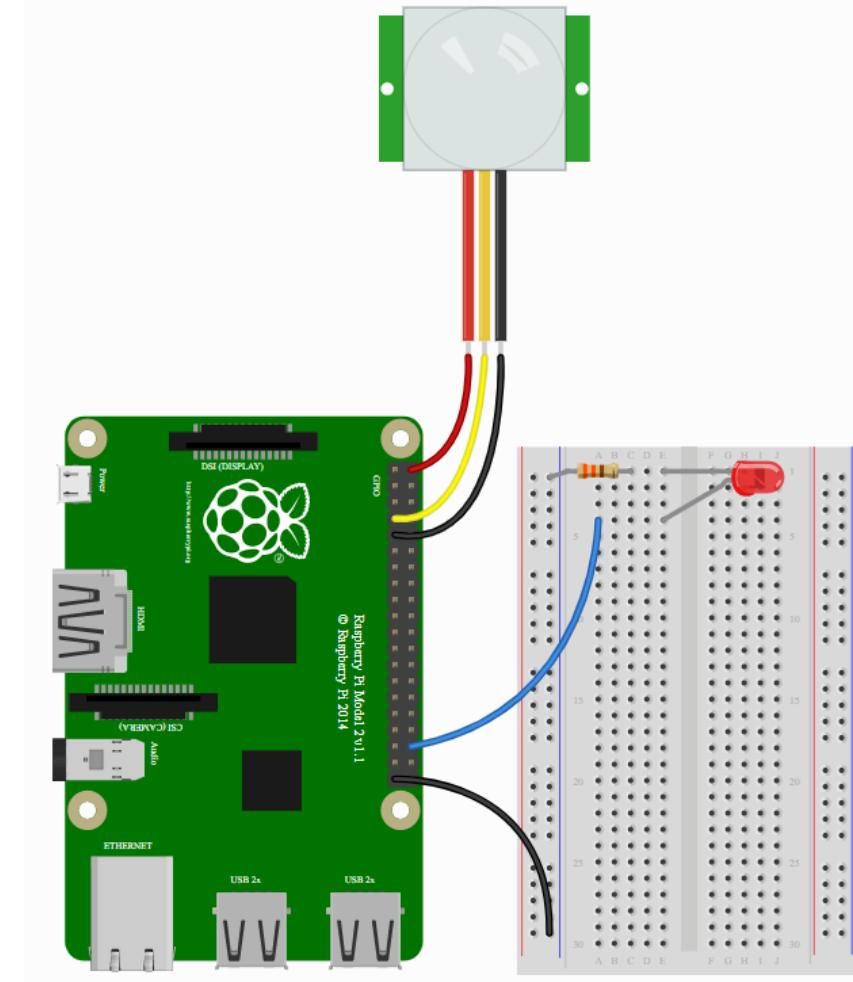
from signal import pause #a função pause é
importada do módulo signal para manter o programa
em execução indefinidamente e responder a eventos
enquanto aguarda uma interrupção

pir = MotionSensor(4) #cria um objeto pir que
representa o sensor de movimento (PIR) na GPIO 4

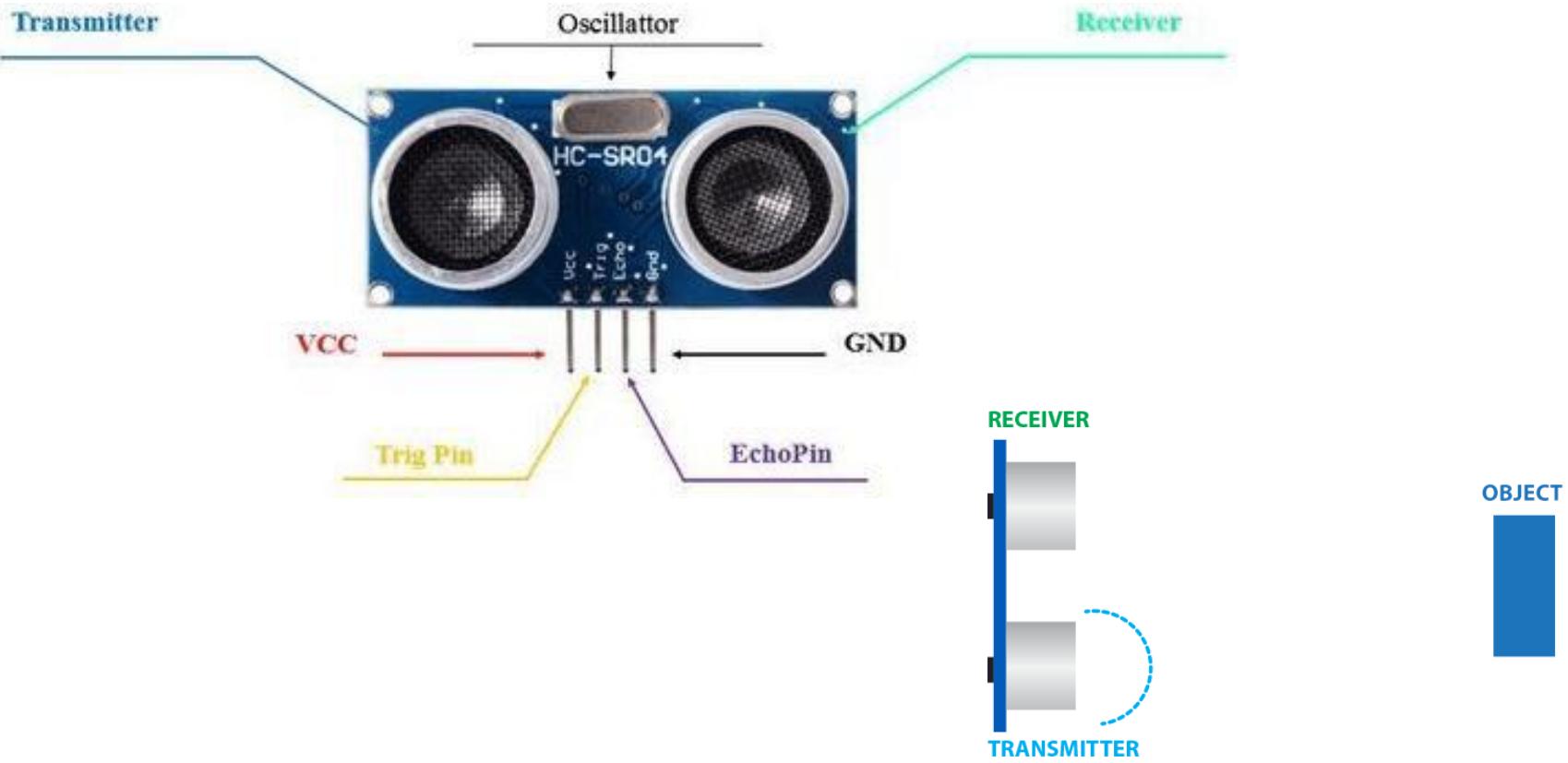
led = LED(16) # objeto LED na GPIO 16
pir.when_motion = led.on # ação para ser executada
quando o sensor de movimento detecta movimento –
liga o LED

pir.when_no_motion = led.off # caso contrário
pause() # aguarda novos eventos

```



Sensor de distância ultrassônico HC-SR04



<linuxhint/>

<https://www.electroschematics.com/hc-sr04-datasheet/>

Sensor de distância ultrassônico HC-SR04

```
from gpiozero import DistanceSensor, LED
from signal import pause

DistanceSensor(echo=24, trigger=23)

led = LED(16)

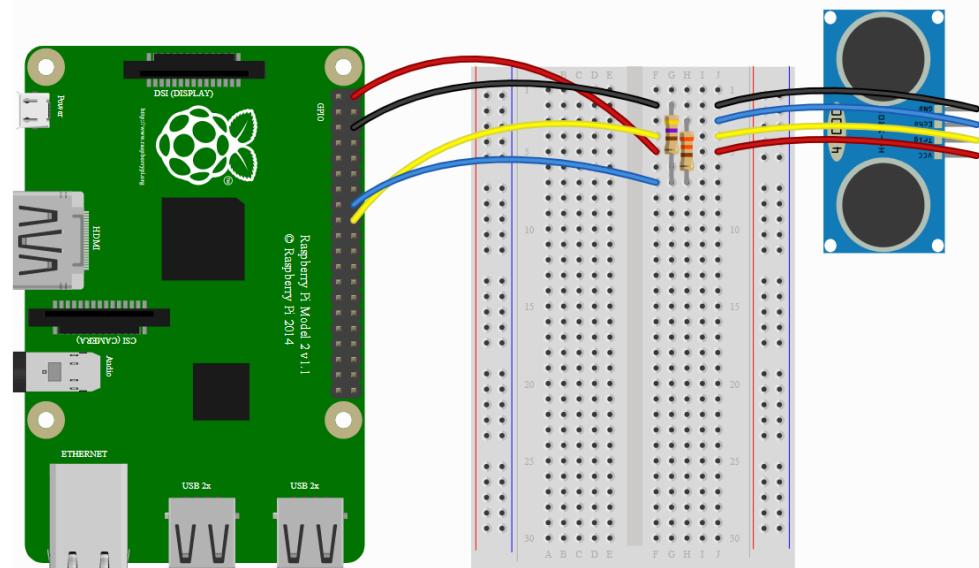
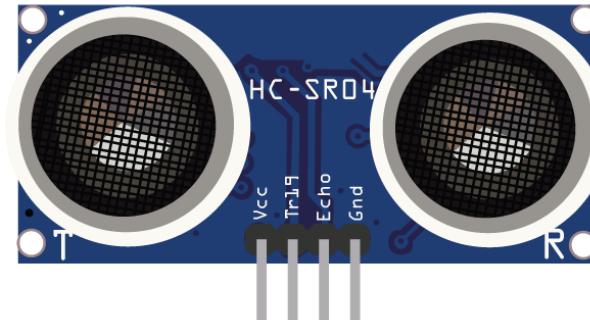
sensor.when_in_range = led.on
sensor.when_out_of_range = led.off

pause()
```

```
from gpiozero import DistanceSensor, LED
import time

DistanceSensor(echo=24, trigger=23)

While True:
    print("distance is:", sensor.distance, "m")
    time.sleep(1)
```



Explorando outros exemplos da documentação

➤ <https://gpiozero.readthedocs.io/en/stable/recipes.html>

□ 2. Basic Recipes

- 2.1. Importing GPIO Zero
- 2.2. Pin Numbering
- 2.3. LED
- 2.4. LED with variable brightness
- 2.5. Button
- 2.6. Button controlled LED
- 2.7. Button controlled camera
- 2.8. Shutdown button
- 2.9. LEDBoard
- 2.10. LEDBarGraph
- 2.11. LEDCharDisplay
- 2.12. Traffic Lights

- 2.17. Full color LED
- 2.18. Motion sensor
- 2.19. Light sensor
- 2.20. Distance sensor
- 2.21. Rotary encoder
- 2.22. Servo
- 2.23. Motors
- 2.24. Robot
- 2.25. Button controlled robot
- 2.26. Keyboard controlled robot
- 2.27. Motion sensor robot
- 2.28. Potentiometer
- 2.29. Measure temperature with an ADC
- 2.30. Full color LED controlled by 3 potentiometers
- 2.31. Timed heat lamp
- 2.32. Internet connection status indicator

3. Advanced Recipes

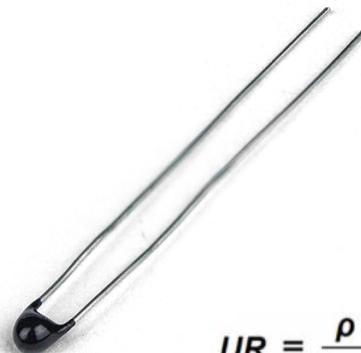
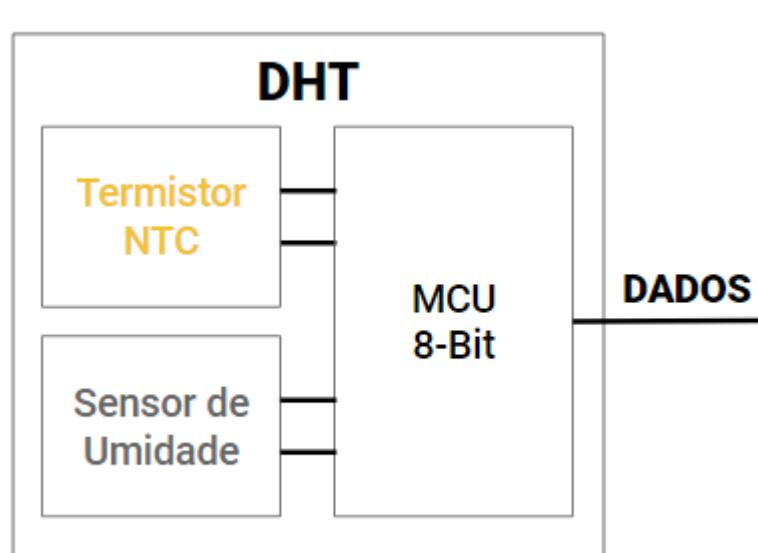
- 3.1. LEDBoard
- 3.2. Multi-character 7-segment display
- 3.3. Who's home indicator
- 3.4. Travis build LED indicator
- 3.5. Button controlled robot
- 3.6. Robot controlled by 2 potentiometers
- 3.7. BlueDot LED
- 3.8. BlueDot robot
- 3.9. Controlling the Pi's own LEDs

Leitura de temperatura e umidade – DHT11

Composto por medidor de umidade (resistência elétrica entre 2 eletrodos – umidade mais elevada diminui a resistência) e um termistor NTC (resistência diminui com aumento da temperatura) para temperatura, ambos conectados a um controlador de 8-bits.

O DHT11 utiliza um protocolo simples para enviar as leituras dos sensores usando apenas um fio de barramento.

- <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>



$$UR = \frac{\rho}{\rho_s} \times 100\%$$

UR: Umidade Relativa

ρ : pressão parcial de vapor de água do ar

ρ_s : pressão de vapor de saturação

Leitura de temperatura e umidade – DHT11

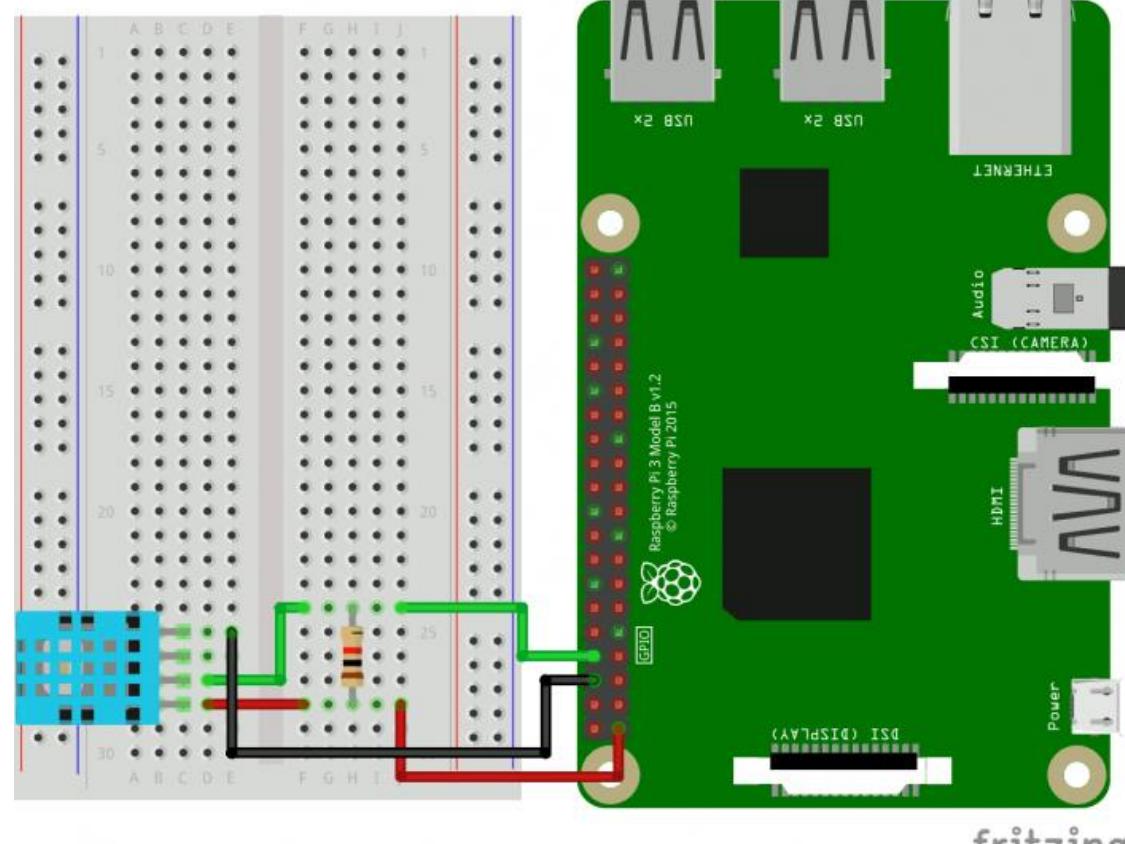
- Fazer o download : https://drive.google.com/file/d/1_peEHSf1XK8PUIMIdxNxOEjPKWQVEbo/view?usp=sharing
- Descompactar a pasta
- No terminal digitar:

```
cd Downloads
```

```
sudo python3 setup.py install
```

```
Shell
Python 3.7.3 (/usr/bin/python3)
>>> %Run leitura_dht_2.py

Temperatura: 30.0 C | Umidade: 16.0 %
Temperatura: 29.0 C | Umidade: 16.0 %
```



Leitura de temperatura e umidade – DHT11

➤ Trecho de configurações no Programa:

✓ Importando a biblioteca

```
import Adafruit_DHT
```

✓ Criação de objeto do sensor usado

```
sensor = Adafruit_DHT.DHT11
```

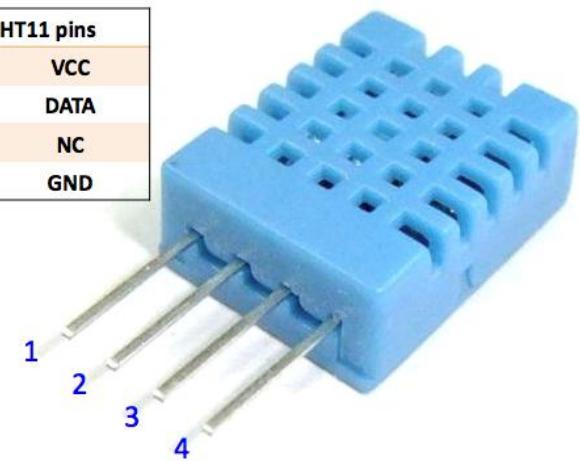
✓ Aquisição de dados de temperatura e umidade por meio da GPIO 9 (conectado ao pino Data do sensor)

```
humidity, temperature = Adafruit_DHT.read_retry(sensor, 9)
```

```
Shell
Python 3.7.3 (/usr/bin/python3)
>>> %Run leitura_dht_2.py

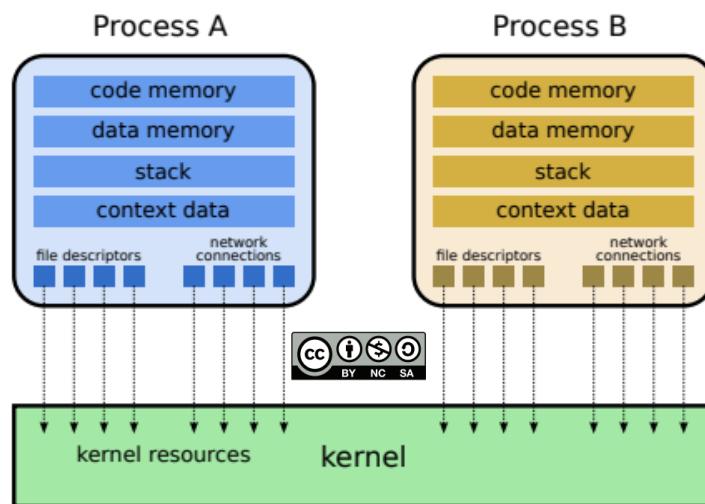
Temperatura: 30.0 C | Umidade: 16.0 %
Temperatura: 29.0 C | Umidade: 16.0 %
```

DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



Processos e threads

- ✓ Um processo é um contêiner de recursos utilizados por uma ou mais tarefas para sua execução: **áreas de memória (código, dados, pilha), informações de contexto e descritores de recursos do núcleo (arquivos abertos, conexões de rede, etc).**
- ✓ Um processo pode então conter várias tarefas, que compartilham esses recursos.
- ✓ Os processos são isolados entre si pelos mecanismos de proteção providos pelo hardware (isolamento de áreas de memória, níveis de operação e chamadas de sistema), impedindo que uma tarefa do processo PA acesse um recurso atribuído ao processo PB.

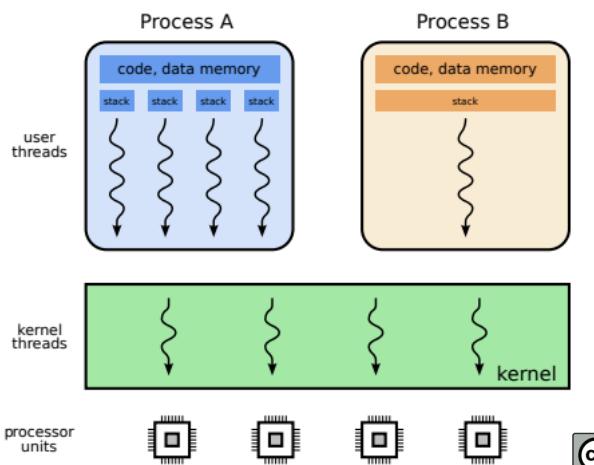


Exemplos de processos:

- ✓ Navegador Web
- ✓ Processamento de imagem e vídeo
- ✓ Reprodutor de mídia
- ✓ Gerenciador de arquivos
- ✓ Processos de inicialização do S.O.
- ✓ Servidor Web
- ✓ Comunicação com dispositivos externos

Processos e threads

- ✓ Uma **thread** é definida como sendo um fluxo de execução independente.
- ✓ Um processo pode conter uma ou mais threads, cada uma executando seu próprio código e compartilhando recursos com as demais **threads** localizadas no mesmo processo.
- ✓ Cada thread é caracterizada por um código em execução e um pequeno contexto local, o chamado **Thread Local Storage (TLS)**, composto pelos registradores do processador e uma área de pilha em memória, para que a **thread** possa armazenar variáveis locais e efetuar chamadas de funções.



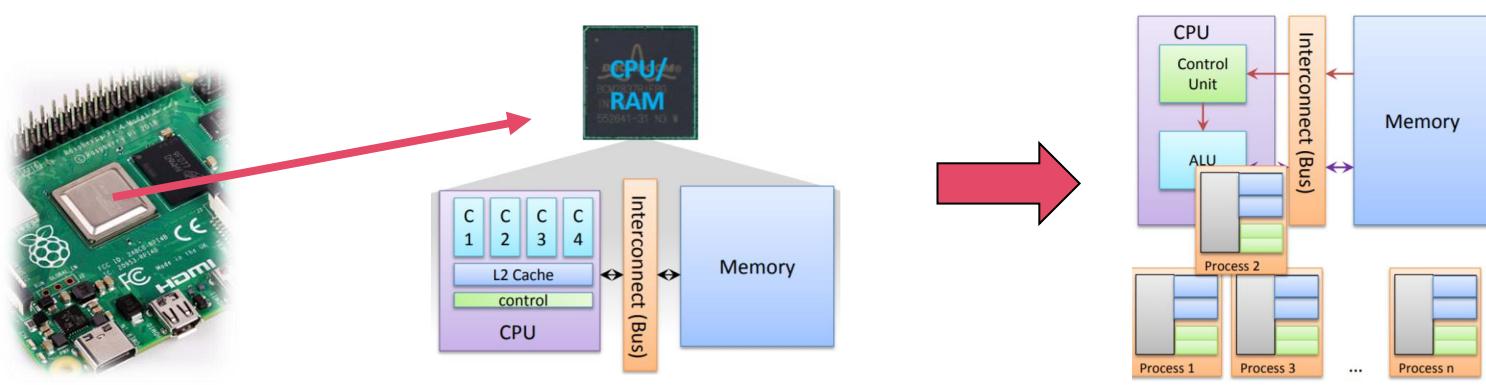
Exemplos de uso threads em processos:

- ✓ Tarefas em background
- ✓ Processamento paralelo de jogos
- ✓ Fila de impressão de documentos
- ✓ Processamento em lotes: edição de imagens, conversão de vídeos, áudio em tempo real



Processos e threads na Raspberry pi

- ✓ **Processos:** qualquer programa rodando em um S.O. possui um ou mais processos associados!
 - ✓ **Thread:** Pode ser definida como uma unidade de execução de um processo. Logo, um processo pode consistir de uma ou várias threads (**multithreading**)!
-
- ✓ **CPU multicore (processador multinúcleo):** possui dois ou mais núcleos (cores) de processamento.
 - ✓ A Raspberry Pi 3B+ ou superior possui arquitetura quad-core (4 núcleos) no seu SoC!
 - ✓ Advém do conceito de **computação paralela**, na qual os núcleos são responsáveis por dividir as tarefas entre eles
 - ✓ Uma CPU multicore com clock de 1,8 GHz significa cada núcleo operando em paralelo com 1,8 GHz.



Trabalhando diretamente com processos no S.O.

- ✓ **Exemplo:** controlando uma saída (LED) conectado à Raspberry Pi e, ao mesmo tempo, realizando uma segunda tarefa aleatória (contagem), em processos separados.

```
import RPi.GPIO as GPIO
import time # para controlar pausas e intervalos de tempo
import random # para gerar números aleatórios.
from multiprocessing import Process # classe Process do módulo
multiprocessing para criar processos paralelos.
import os # para interagir com o sistema operacional, como obter o PID
do processo.

# Configuração do GPIO para piscar o LED
def gpio_blink():
    print(f"Processo de Blink LED iniciado com PID: {os.getpid()}") #
Exibe o PID do processo atual.
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(7, GPIO.OUT)
    while True:
        GPIO.output(7, GPIO.HIGH) # Liga o LED.
        print("LED aceso")
        time.sleep(1)
        GPIO.output(7, GPIO.LOW) # Desliga o LED.
        print("LED apagado")
        time.sleep(1)

# Função para gerar e exibir uma contagem aleatória
def random_count():
    print(f"Processo de Contagem Aleatória iniciado com PID: {os.getpid()}") #
Exibe o PID do processo atual.
    while True:
        count = random.randint(1, 100)
    # Gera um número aleatório entre 1 e 100.
        print(f"Contagem aleatória: {count}")
    # Exibe a contagem aleatória gerada.
        time.sleep(2)
    # Pausa por 2 segundos antes de gerar o próximo número.
```

```
if __name__ == '__main__':
    try:
        # Cria dois processos paralelos
        process1 = Process(target=gpio_blink)
        # Associa a função gpio_blink ao primeiro processo.
        process2 = Process(target=random_count)
        # Associa a função random_count ao segundo processo.

        # Inicia os processos
        process1.start() # Inicia o processo de piscar o LED.
        process2.start() # Inicia o processo de contagem aleatória.

        # Aguarda a conclusão dos processos (o que não ocorre, pois
ambos têm loops infinitos)
        process1.join()
        process2.join()
    except KeyboardInterrupt: # interrupção do teclado (Ctrl+C).
        print("Processos interrompidos.")
    finally:
        GPIO.cleanup() # Limpa a configuração dos pinos GPIO
```

> python process_example.py

Trabalhando diretamente com processos no S.O.

- ✓ Executar o script do slide anterior no terminal: *python script.py* e abrir um segundo terminal para verificar informações sobre o processo relacionado ao programa pelos comandos abaixo

```
> ps aux | grep python #processos específicos relacionados ao script  
> pgrep -fl python #encontrar o ID do processo  
> taskset -cp <PID> # afinidade (núcleos que o processo pode rodar)  
          taskset --cp 2456 # caso o PID do processo seja 2456  
> taskset -cp 2 PID # define a afinidade para núcleo 2  
> top -p <PID> # processo em tempo real  
> ps aux | grep <PID> ou pgrep -f script.py #mais informações sobre o processo  
          #outros comandos: htop, ps aux, pstree etc  
> python script.py & # o & coloca o processo em segundo plano  
> kill <PID> #finaliza o processo
```

Quando usar threads

O código ao lado é sequencial, ou seja, executa `gpio_blink()` e, depois, `random_count()`. Como as funções têm loops infinitos, o código não sairá dos loops.

Ao executar esse programa no terminal, é provável que somente a função `gpio_blink` seja executada!

Em Python, o código dentro de uma função em loop não permitirá a execução de outras funções simultaneamente se não houver mecanismos para gerenciar a execução paralela, como threads ou processos.

Cada função é executada uma após a outra no mesmo fluxo de controle (o código é executado linha após linha).

Se a primeira função entra em um loop infinito, o controle nunca alcança a próxima função.

```
def primeira_funcao():
    while True:
        print("Executando a primeira função")

def segunda_funcao():
    print("Executando a segunda função")

if __name__ == '__main__':
    primeira_funcao() # Esta função entra em um loop infinito
    segunda_funcao() # Esta linha nunca será alcançada
```

```
import RPi.GPIO as GPIO
import time
import random

# Configuração do GPIO para piscar o LED
def gpio_blink():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(7, GPIO.OUT) # Usando o pino 7 para o LED

    while True:
        GPIO.output(7, GPIO.HIGH)
        print("LED aceso") # Mensagem quando o LED está aceso
        time.sleep(1) # LED aceso por 1 segundo
        GPIO.output(7, GPIO.LOW)
        print("LED apagado") # Mensagem quando o LED está apagado
        time.sleep(1) # LED apagado por 1 segundo

# Contagem aleatória
def random_count():
    while True:
        count = random.randint(1, 100)
        print(f"Contagem aleatória: {count}")
        time.sleep(2) # Pausa de 2 segundos entre contagens

if __name__ == '__main__':
    try:
        # Inicia as funções para piscar o LED e contar aleatoriamente
        gpio_blink() # Piscar LED
        random_count() # Contagem aleatória
    except KeyboardInterrupt:
        print("Interrompido pelo usuário.")
    finally:
        GPIO.cleanup()
```

Quando usar threads

A solução é usar threads para executar `gpio_blink` e `random_count` simultaneamente!

As threads garantem que o programa execute tarefas em paralelo, evitando que uma função com um loop infinito bloqueeie a execução das outras.

Usar os comandos **ps**, **top**, **htop**, **pstree**, e **ps -o pid,tid**, e **comm** no terminal (abrir um segundo terminal enquanto executa o programa no primeiro) para detalhes sobre as threads/processos em execução

```
import threading

def primeira_funcao():
    while True:
        print("Executando a primeira função")

def segunda_funcao():
    while True:
        print("Executando a segunda função")

if __name__ == '__main__':
    # Cria e inicia as threads
    thread1 = threading.Thread(target=primeira_funcao)
    thread2 = threading.Thread(target=segunda_funcao)

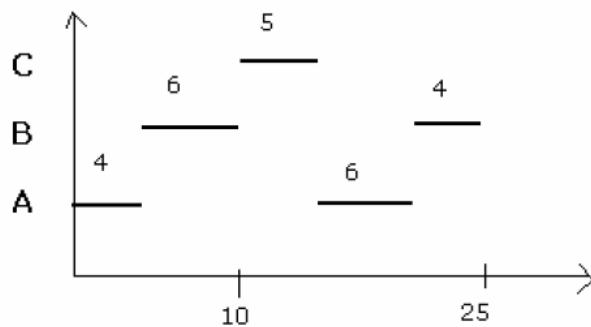
    thread1.start() # Inicia a execução da primeira função em uma nova thread
    thread2.start() # Inicia a execução da segunda função em outra nova thread

    thread1.join() # Aguarda a conclusão da primeira thread
    thread2.join() # Aguarda a conclusão da segunda thread
```

Round-robin e Preempção

- Round-robin é um algoritmo de escalonamento que distribui o tempo de CPU de forma circular entre os processos ou threads. Cada processo ou thread recebe um intervalo de tempo fixo, conhecido como "quantum" (quando ele esgota, o próximo processo ou thread na fila é selecionado).

No exemplo, cria-se múltiplos processos que executam por um curto período e monitorar como o sistema operacional alterna entre eles (verificar com **top** e **htop**)



```

import time
import threading

def task(name, duration):
    start_time = time.time()
    while time.time() - start_time < duration:
        print(f"{name} está executando por {time.time() - start_time:.2f} segundos")
        time.sleep(0.5)

def main():
    duration = 5 # Tempo de execução para cada tarefa
    thread1 = threading.Thread(target=task, args=("Thread 1", duration))
    thread2 = threading.Thread(target=task, args=("Thread 2", duration))
    thread3 = threading.Thread(target=task, args=("Thread 3", duration))

    thread1.start()
    thread2.start()
    thread3.start()

    thread1.join()
    thread2.join()
    thread3.join()

if __name__ == "__main__":
    main()

```

Round-robin e Preempção

- **Preempção** é um mecanismo que permite que o sistema operacional interrompa um processo ou thread em execução para dar tempo de CPU a outro processo ou thread, garantindo que todos recebam uma fatia justa de tempo do microprocessador. Em sistemas multitarefa, essa ação garante que nenhum processo monopolize a CPU.

No exemplo, cria-se dois processos que fazem uso da CPU. O sistema operacional deve alternar entre esses processos para garantir que ambos recebam tempo do processador (verificar com comandos **top** e **htop**)

```
import time
import threading

def cpu_intensive_task(name):
    while True:
        print(f"{name} está consumindo CPU")
        # Simula trabalho intenso de CPU
        _ = sum(x * x for x in range(10000))

def main():
    thread1 = threading.Thread(target=cpu_intensive_task, args=("Thread 1",))
    thread2 = threading.Thread(target=cpu_intensive_task, args=("Thread 2",))

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()

if __name__ == "__main__":
    main()
```

Lembrando que o sentido de `join()` (tanto em processos como threads) é forçar o código principal a aguardar a conclusão do código paralelo, "reunindo" o controle de volta para o fluxo de execução principal

Rastreando threads

- Usando threads para executar gpio_blink e random_count simultaneamente! Tanto o piscar do LED quanto a contagem aleatória devem ser exibidos simultaneamente no terminal. Os prints que mostram o PID do processo principal e o identificador da thread (ps -o pid,tid,comm: exibem o PID e o TID (Thread ID)).
- **Exemplo: ps -eLf | grep <PID>**

```
import RPi.GPIO as GPIO
import time
import random
import threading
import os

# Configuração do GPIO para piscar o LED
def gpio_blink():
    print(f"Thread para piscar LED iniciada com ID: {threading.get_ident()}")
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(7, GPIO.OUT) # Usando o pino 7 para o LED

    while True:
        GPIO.output(7, GPIO.HIGH)
        print("LED aceso") # Mensagem quando o LED está aceso
        time.sleep(1) # LED aceso por 1 segundo
        GPIO.output(7, GPIO.LOW)
        print("LED apagado") # Mensagem quando o LED está apagado
        time.sleep(1) # LED apagado por 1 segundo
```

```
# Contagem aleatória
def random_count():
    print(f"Thread de contagem aleatória iniciada com ID: {threading.get_ident()}")
    while True:
        count = random.randint(1, 100)
        print(f"Contagem aleatória: {count}")
        time.sleep(2) # Pausa de 2 segundos entre contagens

if __name__ == '__main__':
    try:
        print(f"Processo principal PID: {os.getpid()}") # Exibe o PID do processo principal

        # Cria threads para executar as funções simultaneamente
        thread1 = threading.Thread(target=gpio_blink)
        thread2 = threading.Thread(target=random_count)

        # Inicia as threads
        thread1.start()
        thread2.start()

        # Aguarda a conclusão das threads
        thread1.join()
        thread2.join()
    except KeyboardInterrupt:
        print("Interrompido pelo usuário.")
    finally:
        GPIO.cleanup()
```

Mutex e semáforos

- Exemplo básico onde várias threads tentam acessar e modificar um recurso compartilhado sem qualquer controle de sincronização.

Problema: sem sincronização, o valor final de contagem pode ser incorreto devido a condições de corrida*, onde as threads podem ler e escrever o valor simultaneamente sem coordenação.

```
import threading
import time

# Recurso compartilhado
contagem = 0

def incrementar():
    global contagem
    for _ in range(100000):
        contagem += 1
    print(f"Contagem final: {contagem}")

if __name__ == '__main__':
    thread1 = threading.Thread(target=incrementar)
    thread2 = threading.Thread(target=incrementar)

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()

    print(f"Contagem final após todas as threads: {contagem}")
```

* Apesar disso, o Global Interpreter Lock (GIL) do Python pode impedir condições de corrida, evitando o problema relatado acima (portanto, pode ser que a contagem aconteça corretamente)

Mutex e semáforos

- Mutex (Mutual Exclusion) é um mecanismo de sincronização que garante que apenas uma thread possa acessar o recurso compartilhado de cada vez.

Apenas uma thread de cada vez pode acessar e modificar contagem, evitando condições de corrida

- Usar Mutex ou Semáforo:

Mutex: garantir a exclusão mútua e que apenas uma thread acesse o recurso por vez (quando o recurso é crítico e não deve ser acessado simultaneamente por múltiplas threads).

Semáforo: permitir que um número limitado de threads acesse um recurso ao mesmo tempo.

```
import threading
import time

# Recurso compartilhado
contagem = 0

mutex = threading.Lock() # Cria um mutex

def incrementar():
    global contagem
    for _ in range(100000):
        with mutex: # Adquire o mutex antes de acessar o recurso
            contagem += 1
    print(f"Contagem final: {contagem}")

if __name__ == '__main__':
    thread1 = threading.Thread(target=incrementar)
    thread2 = threading.Thread(target=incrementar)

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()

    print(f"Contagem final após todas as threads: {contagem}")
```

Mutex e semáforos

- **Semáforo** é um mecanismo de sincronização que controla o acesso a um recurso compartilhado, permitindo que um número específico de threads acesse o recurso ao mesmo tempo

O semáforo aqui é configurado com um valor inicial de 1, o que o torna equivalente a um mutex. No entanto, semáforos podem ser configurados com valores maiores para permitir que várias threads acessem o mesmo recurso simultaneamente.

* Lembrando que o interpretador GIL do Python pode atuar como um mutex, fazendo por si só com que somente uma thread seja executada por vez

```
import threading
import time

# Recurso compartilhado
contagem = 0
semaforo = threading.Semaphore(1) # Semáforo com valor inicial 1 (sempre 1)

def incrementar():
    global contagem
    for _ in range(100000):
        with semaforo: # Adquire o semáforo antes de acessar o recurso
            contagem += 1
    print(f"Contagem final: {contagem}")

if __name__ == '__main__':
    thread1 = threading.Thread(target=incrementar)
    thread2 = threading.Thread(target=incrementar)

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()

    print(f"Contagem final após todas as threads: {contagem}")
```

Deadlock

- **Deadlock** ocorre quando duas ou mais threads ficam bloqueadas indefinidamente, esperando por um recurso que é possuído por outra thread

Thread 1 adquire mutex1 e tenta adquirir mutex2.

Thread 2 adquire mutex2 e tenta adquirir mutex1.

Se thread1 adquirir mutex1 e thread2 adquirir mutex2 quase ao mesmo tempo, ambas ficarão bloqueadas esperando uma pela outra, resultando em deadlock.

```
import threading
import time

# Recurso compartilhado
mutex1 = threading.Lock()
mutex2 = threading.Lock()

def tarefa1():
    with mutex1:
        print("Thread 1 adquiriu mutex1")
        time.sleep(1) # Simula algum trabalho
    with mutex2:
        print("Thread 1 adquiriu mutex2")

def tarefa2():
    with mutex1:
        print("Thread 2 adquiriu mutex1")
        time.sleep(1) # Simula algum trabalho
    with mutex2:
        print("Thread 2 adquiriu mutex2")

if __name__ == '__main__':
    thread1 = threading.Thread(target=tarefa1)
    thread2 = threading.Thread(target=tarefa2)

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()
```

Deadlock – solução

- A solução é garantir que todas as threads adquiram os mutexes na mesma ordem. Dessa forma, evitar-se-á o cenário em que uma thread possui um mutex e a outra thread possui o mutex que a primeira precisa.

As duas funções (tarefa1 e tarefa2) agora adquirem mutex1 antes de mutex2, o que evita que uma thread possa adquirir um mutex e a outra thread o mutex que a primeira solicitou, o que evitará o deadlock.

➤ Como evitar deadlocks:

- ✓ Adquirir recursos em uma ordem fixa e hierárquica.
- ✓ Usar timeouts para evitar que uma thread fique bloqueada indefinidamente.
- ✓ Minimizar a necessidade de adquirir múltiplos locks e preferir uma abordagem usando apenas um lock

```
import threading
import time

# Recurso compartilhado
mutex1 = threading.Lock()
mutex2 = threading.Lock()

def tarefa1():
    with mutex1:
        print("Thread 1 adquiriu mutex1")
        time.sleep(1) # Simula algum trabalho
    with mutex2:
        print("Thread 1 adquiriu mutex2")

def tarefa2():
    with mutex1:
        print("Thread 2 adquiriu mutex1")
        time.sleep(1) # Simula algum trabalho
    with mutex2:
        print("Thread 2 adquiriu mutex2")

if __name__ == '__main__':
    thread1 = threading.Thread(target=tarefa1)
    thread2 = threading.Thread(target=tarefa2)

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()
```

Referências e créditos

- Computação Científica em Python- Prof. Luiz T. F. Eleno. EEL-USP; DEMAR –LOM3260. Disponível em: <https://computeel.org/LOM3260/>
- GPIO Zero: <https://gpiozero.readthedocs.io/en/stable/index.html>
- Nunes, A. H. D.; Amaral, I. F. Learning. Py. PETEE UFMG. Universidade Federal de Minas Gerais. Recurso digital. 2020
- Operating Systems Foundations with Linux on the Raspberry Pi - ARM Education Media.
- Pinout – The Raspberry Pi Pinout Guide . Disponível em: <https://pinout.xyz>
- Portal Embarcados- <https://embarcados.com.br>
- Python – <https://www.python.org>.
- Raspberry Pi Foundation - <https://www.raspberrypi.org> - <https://www.raspberrypi.com>