Internet of Things 420-420-LE

Week 5: The GPIO interface

CHAMPLAIN COLLEGE

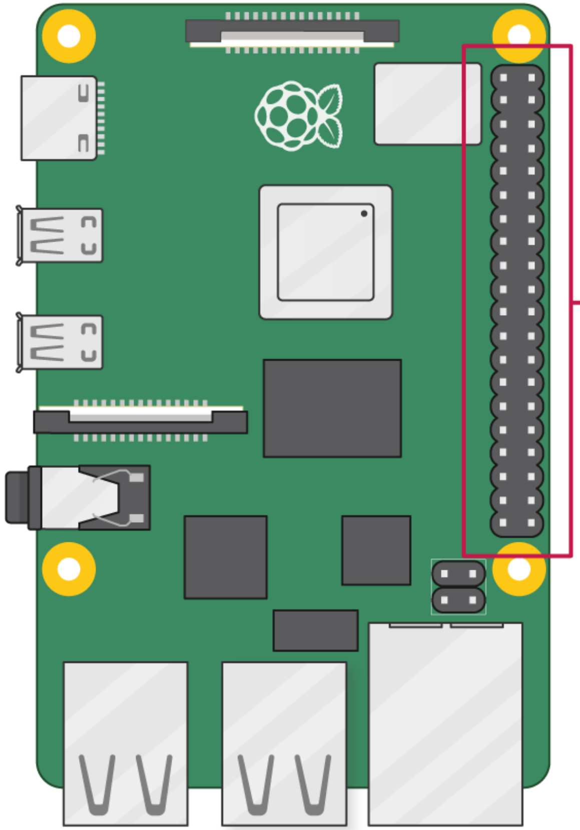
Rev.: Jan. 15th, 2025

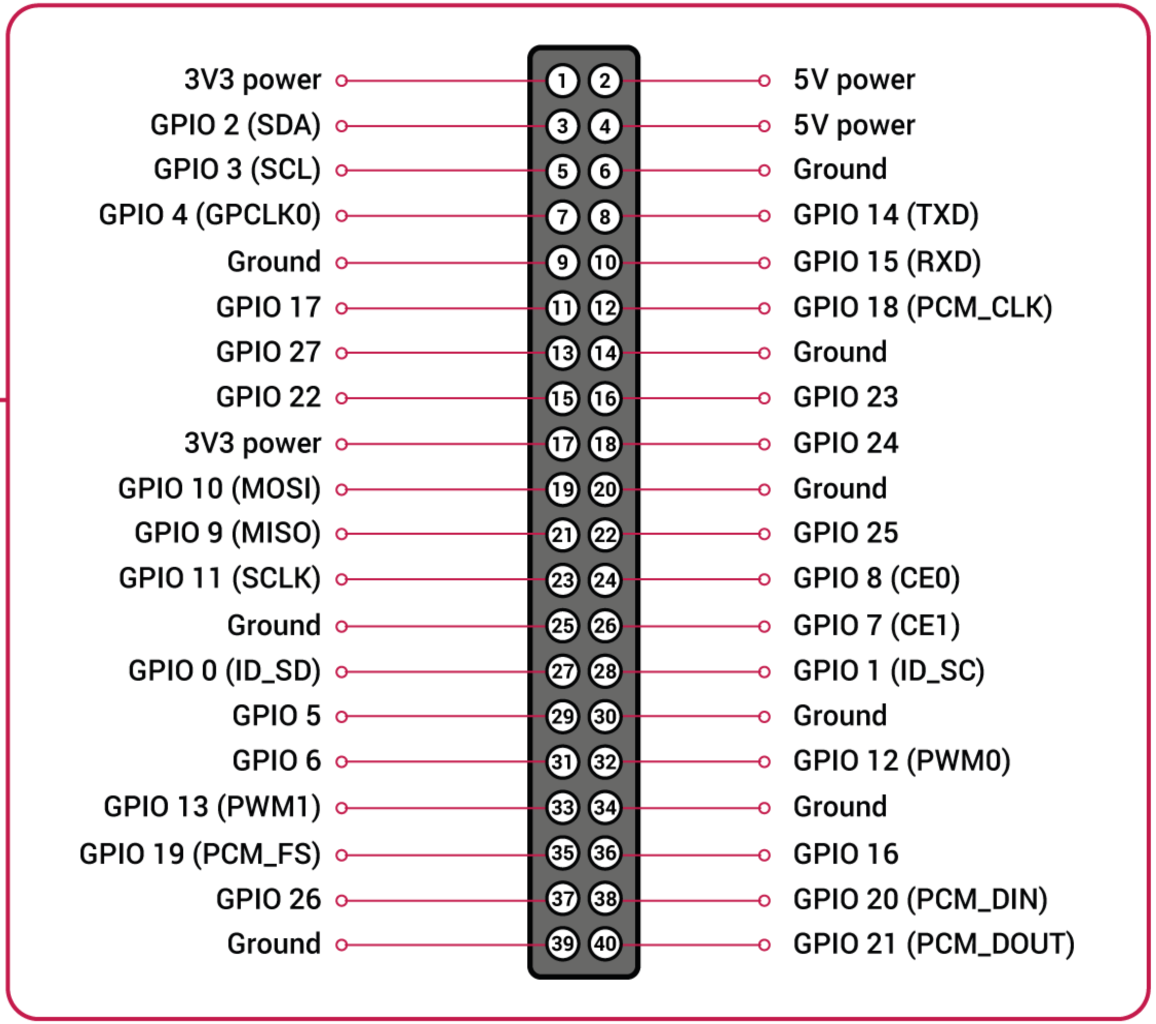
The GPIO Interface

GPIO

- A row of GPIO (**General Purpose Input/Output**) pins along the top edge of the board.
- At the simplest level, you can think of them as switches that the Pi can turn on or off (output) or that you can turn on or off (input).
- The GPIO pins allow the Raspberry Pi to control and monitor the outside world by being connected to electronic circuits.
- A 40 pin GPIO header is found on all current Raspberry Pi. Prior to the Pi 1 Model B+ (2014), boards comprised a shorter 26 pin header.

GPIO

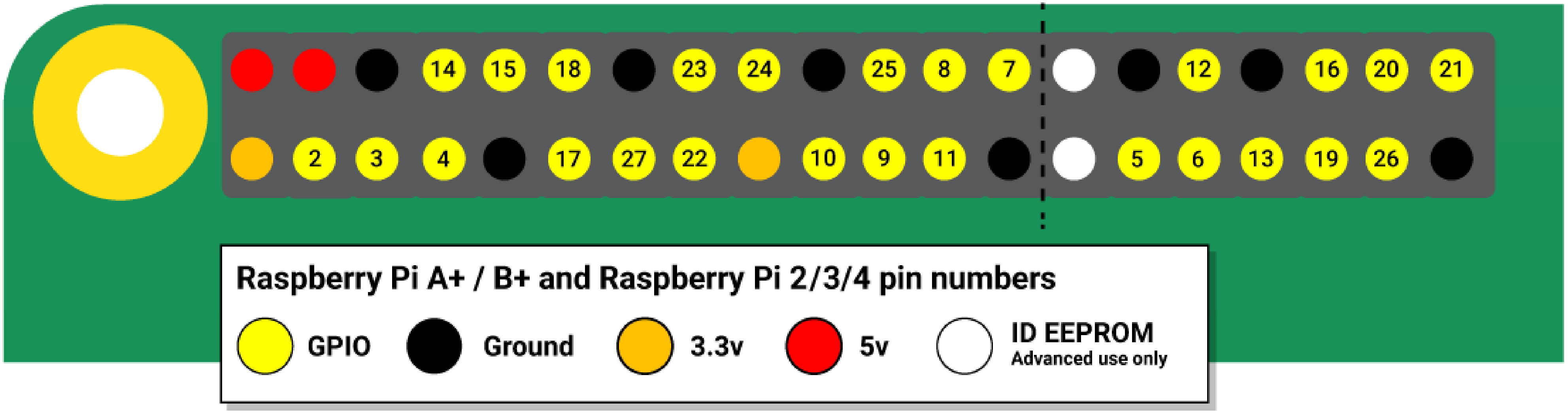




3V3 power
GPIO 2 (SDA)
GPIO 3 (SCL)
GPIO 4 (GPCLK0)
Ground
GPIO 17
GPIO 27
GPIO 22
3V3 power
GPIO 10 (MOSI)
GPIO 9 (MISO)
GPIO 11 (SCLK)
Ground
GPIO 0 (ID_SD)
GPIO 5
GPIO 6
GPIO 13 (PWM1)
GPIO 19 (PCM_FS)
GPIO 26
Ground
1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
17 18
19 20
21 22
23 24
25 26
27 28
29 30
31 32
33 34
35 36
37 38
39 40
5V power
5V power
Ground
GPIO 14 (TXD)
GPIO 15 (RXD)
GPIO 18 (PCM_CLK)
Ground
GPIO 23
GPIO 24
Ground
GPIO 25
GPIO 8 (CE0)
GPIO 7 (CE1)
GPIO 1 (ID_SC)
Ground
GPIO 12 (PWM0)
Ground
GPIO 16
GPIO 20 (PCM_DIN)
GPIO 21 (PCM_DOUT)

GPIO

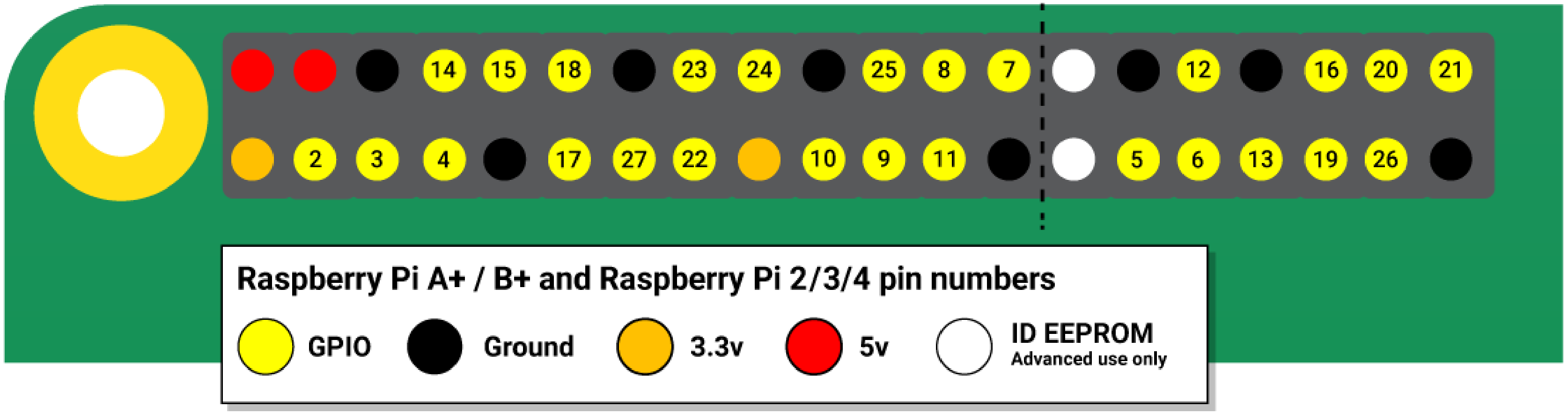
- Any of the GPIO pins can be designated (in software) as an input or output pin and used for a wide range of purposes.



GPIO pin types

Voltages

- Two 5V pins and two 3.3V pins are present on the board, as well as several ground pins (0V), which are **unconfigurable**.
- The remaining pins are all general purpose 3.3V pins, meaning outputs are set to 3.3V and inputs are 3.3V **tolerant**.



GPIO pin types

Outputs

- A GPIO pin designated as an output pin can be set to high (3.3V) or low (0V).

Inputs

- A GPIO pin designated as an input pin can be read as high (3.3V) or low (0V).

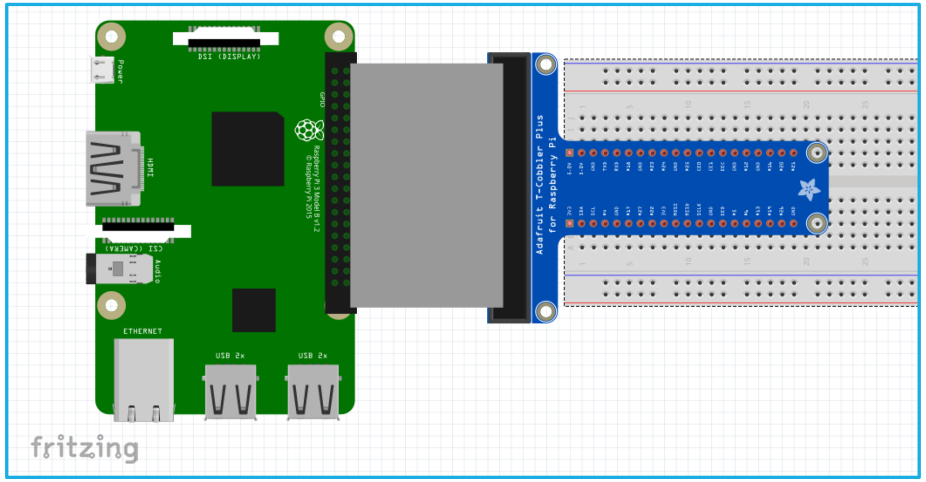
GPIO pin types, more…

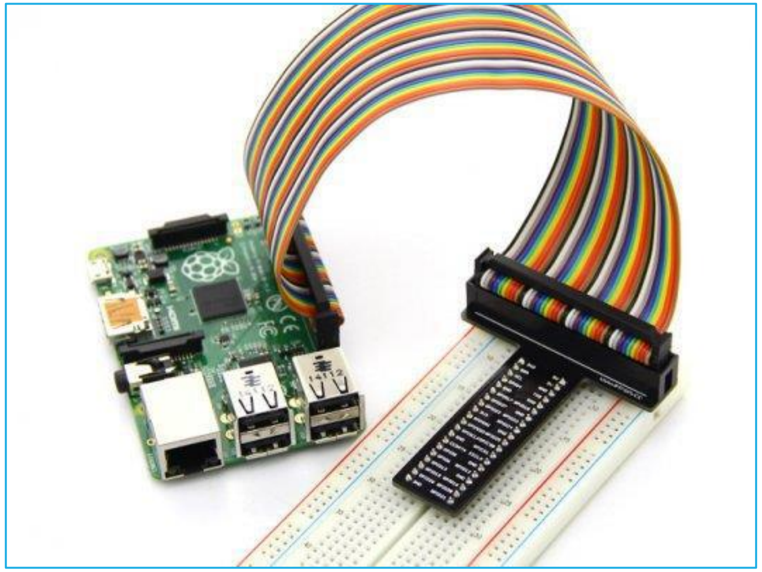
As well as simple input and output devices, the GPIO pins can be used with a variety of alternative functions, some are available on all pins, others on specific pins.

- PWM (Pulse Width Modulation)
  - Software PWM available on all pins
  - Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
- SPI (Serial Peripheral Interface)
  - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1
  - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- I2C (Inter Integrated Circuit)
  - Data: (GPIO2); Clock (GPIO3)
  - EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
- Serial
  - TX (GPIO14); RX (GPIO15)

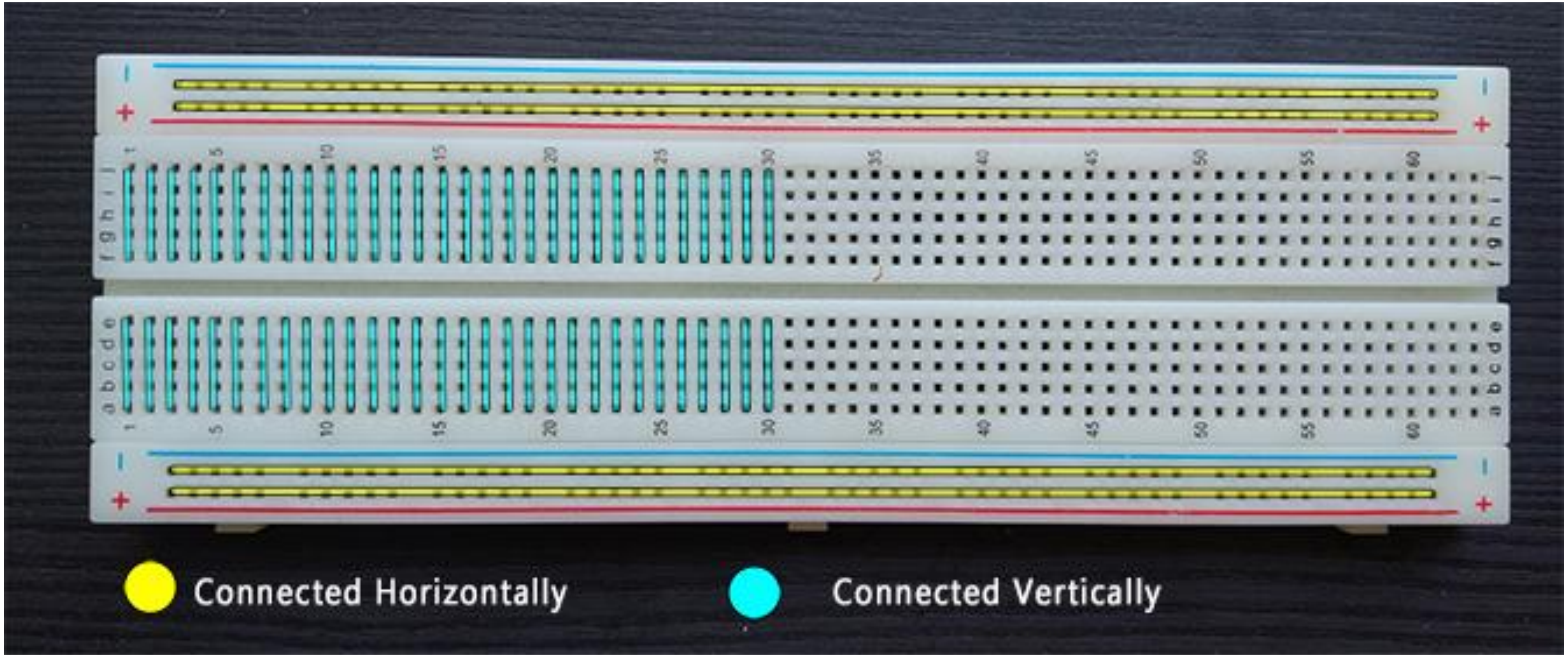
Setting up the hardware

- Before you can dive into coding, you need to set up the hardware environment for the project.
- You need to use the T-Cobbler breakout board and the breadboard



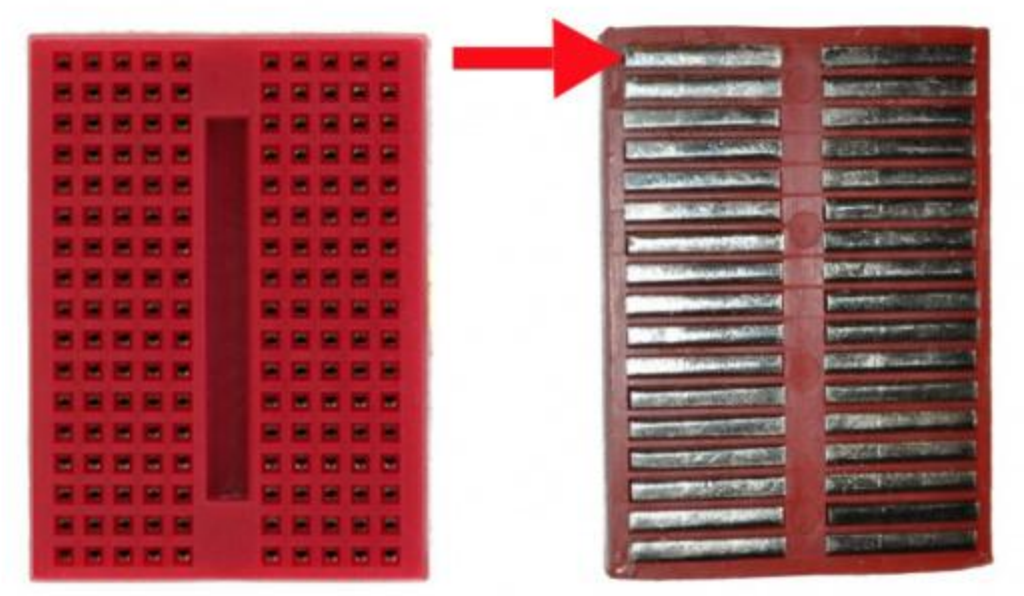


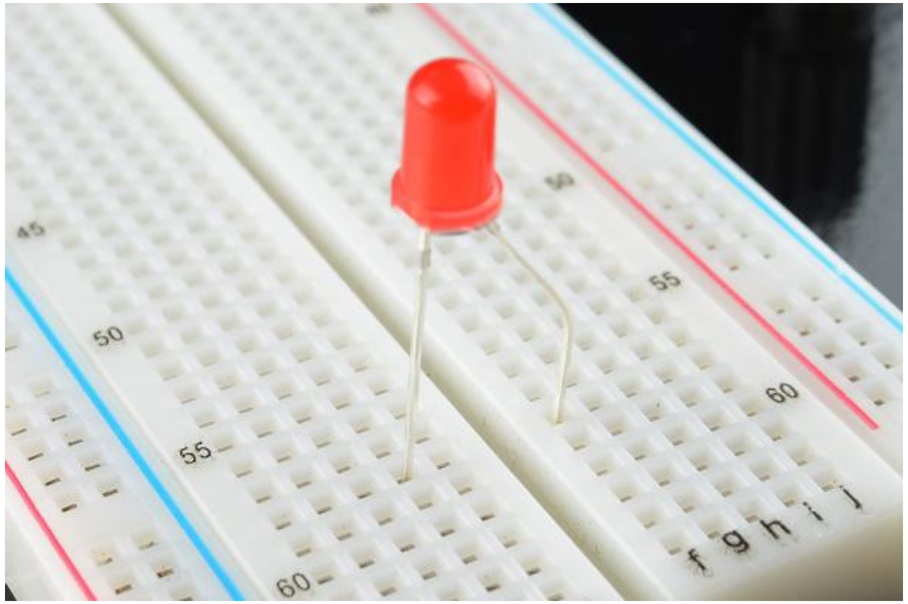
Anatomy of a Breadboard



Terminal Strips

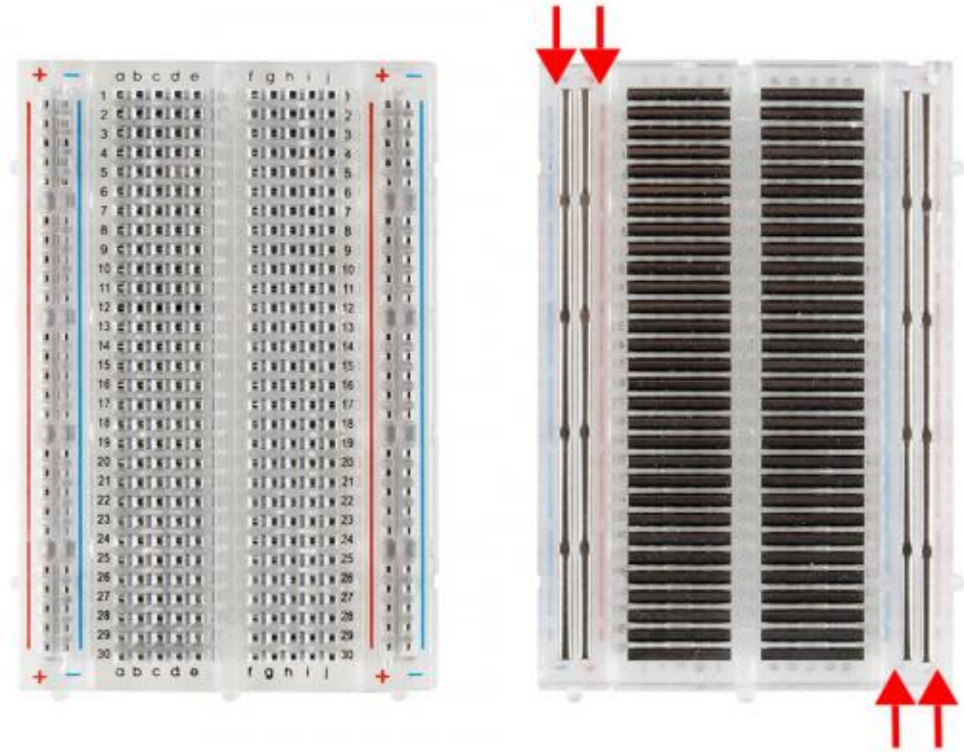
- Here we have a breadboard where the adhesive backing has been removed. You can see lots of horizontal rows of metal strips on the bottom of the breadboard.





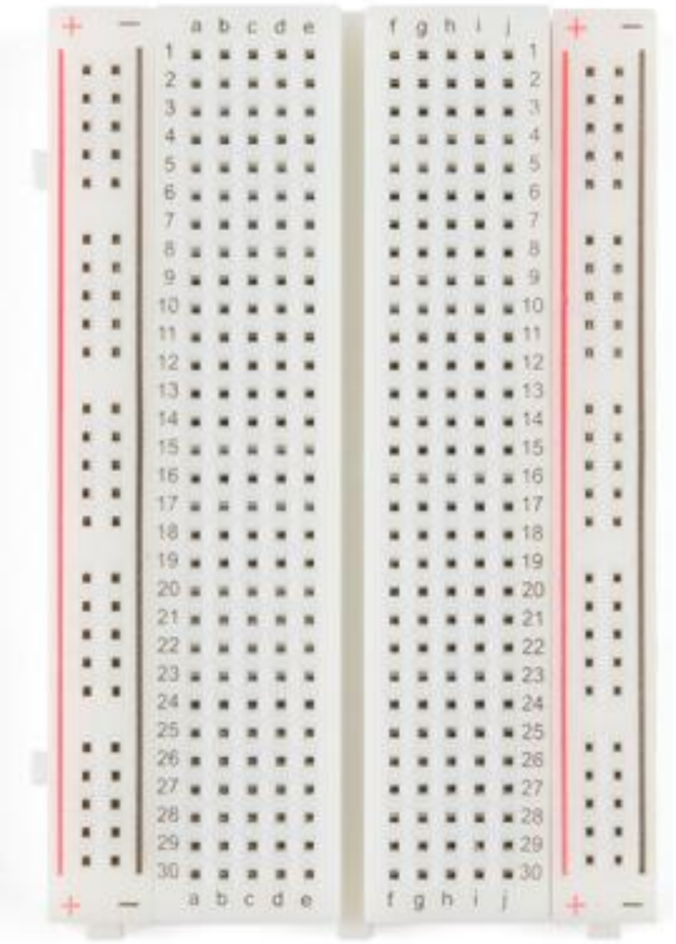
Power Rails

- They are metal strips that are identical to the ones that run horizontally, except they run vertically
- The power rails give you lots of easy access to power wherever you need it in your circuit.
- Usually, they are labeled with a ‘+’ and a ‘-’ and have a red and blue or black stripe, to indicate the positive and negative side



Rows and Columns

- Many breadboards have numbers and letters marked on various rows and columns.
- These don't serve any purpose other than to help guide you when building your circuit.
- If you know the row number of the connection you are trying to make, it makes it much simpler to plug a wire into that number rather than eyeballing it.



Using RPi.GPIO

Using Rpi.GPIO

- To interface your Python programs with the GPIO signals, you must use the RPi.GPIO module.
- The RPi.GPIO module uses direct memory access to provide an interface to control the GPIO signals.
- RPi.GPIO documentation:

  https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/

Validating Rpi.GPIO

- If the module RPi.GPIO is well installed, you can use the `pinout` command to get some information about your GPIO system.

```
gastudillo@gastudillo:~ $ pinout
,--------------------------------.
| oooooooooooooooooooo J8   +======
| 1ooooooooooooooooooo  PoE |   Net
|  Wi                    1o +======
|  Fi  Pi Model 4B  V1.2 oo      |
|        ,----. +---+         +====
| |D|    |SoC | |RAM|         |USB3
| |S|    |    | |   |         +====
| |I|    `----' +---+            |
|                   |C|       +====
|                   |S|       |USB2
| pwr   |hd|   |hd| |I||A|    +====
`-| |---|m0|---|m1|----|V|-------'

Revision           : c03112
SoC                : BCM2711
RAM                : 4GB
Storage            : MicroSD
USB ports          : 4 (of which 2 USB3)
Ethernet ports     : 1 (1000Mbps max. speed)
Wi-fi              : True
Bluetooth          : True
Camera ports (CSI) : 1
Display ports (DSI): 1

J8:
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (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
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
```

Startup methods

- Before you can start interacting with the interface, you have to use the `setmode()` method to set how the library will reference the GPIO pins.
- There are two ways to reference the GPIO signals:
  - Using the pin number on the GPIO interface
  - Using the GPIO signal number from the Broadcom chip

GPIO.BOARD vs GPIO.BCM

- The `GPIO.BOARD` option, tells the library to reference signals based on the pin number on the GPIO interface:

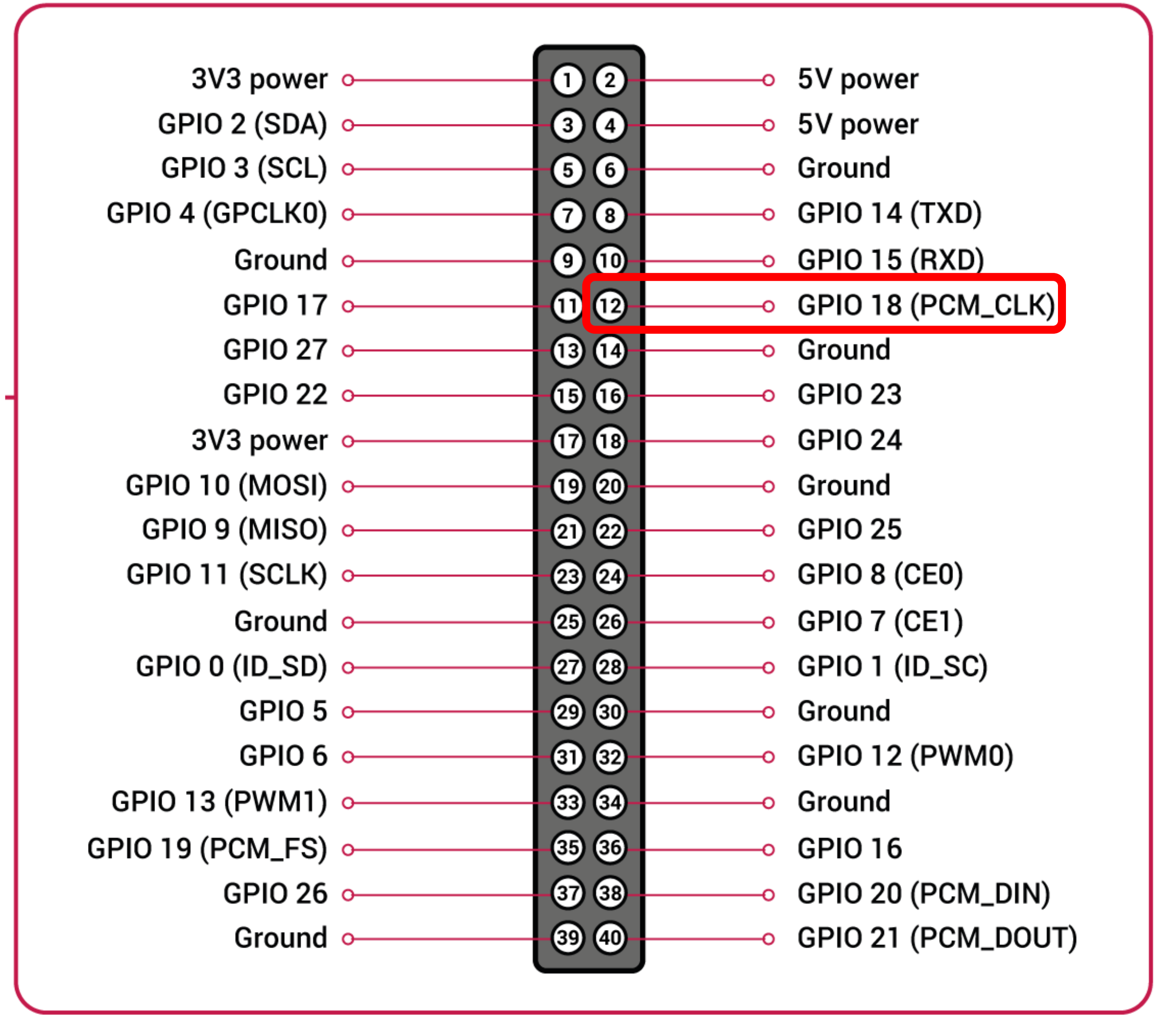
```
GPIO.setmode(GPIO.BOARD)
```

- We can also use the Broadcom chip signal number, specified by the GPIO.BCM value:

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

GPIO.BOARD vs GPIO.BCM

- For example, GPIO signal 18 is on pin 12 of the GPIO interface.
- If you use the GPIO.BCM mode, you reference it using the number 18.
- If you use the GPIO.BOARD mode, you reference it using the number 12.



Startup Methods

- After you select the mode, you must define which GPIO signal to use in your program and whether it will be used for input or output.
- You can do with the setup() method:

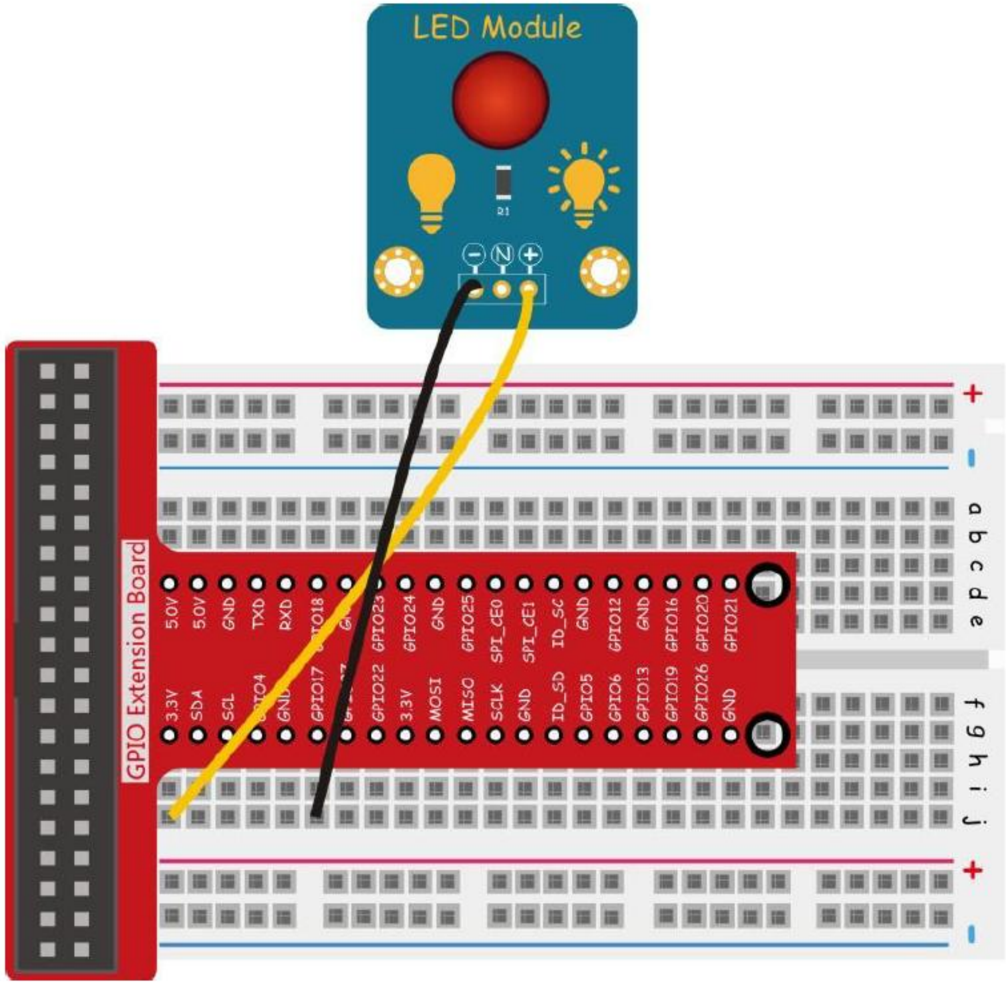
```
GPIO.setup(channel, direction)
```

- For the direction parameter, you can use constants defined in the library: GPIO.IN and GPIO.OUT.
- For example, to set GPIO signal 18 as ***output***:

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

Controlling the GPIO output

Testing the GPIO Output



Type these lines in the Python interpreter one at time:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(17, GPIO.LOW)
GPIO.setup(17, GPIO.HIGH)
GPIO.setup(17, GPIO.LOW)
GPIO.setup(17, GPIO.HIGH)
GPIO.cleanup()
```

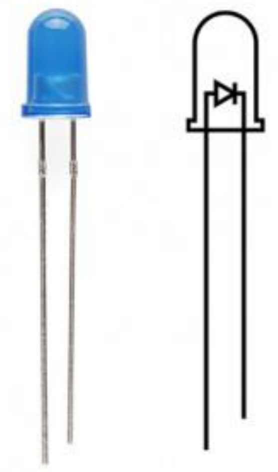
Correct use of GPIO.cleanup()

- RPi.GPIO provides a built-in function GPIO.cleanup() to clean up all the ports you've used.
- It only affects any ports you have set in the current program.
- It resets any ports you have used in this program back to ***input mode***.
- This prevents damage from a situation where you have a port set HIGH as an output and you accidentally connect it to GND (LOW), which would short-circuit the port and possibly fry it.
- Inputs can handle either 0V (LOW) or 3.3V (HIGH), so it's safer to leave ports as inputs.

Blinking a LED

What is a LED?

- LED is the abbreviation of light emitting diode. It is usually made of semiconductor materials.
- Current flows from the anode to the cathode and never the opposite direction.
- The color of light depends on the materials it was made.

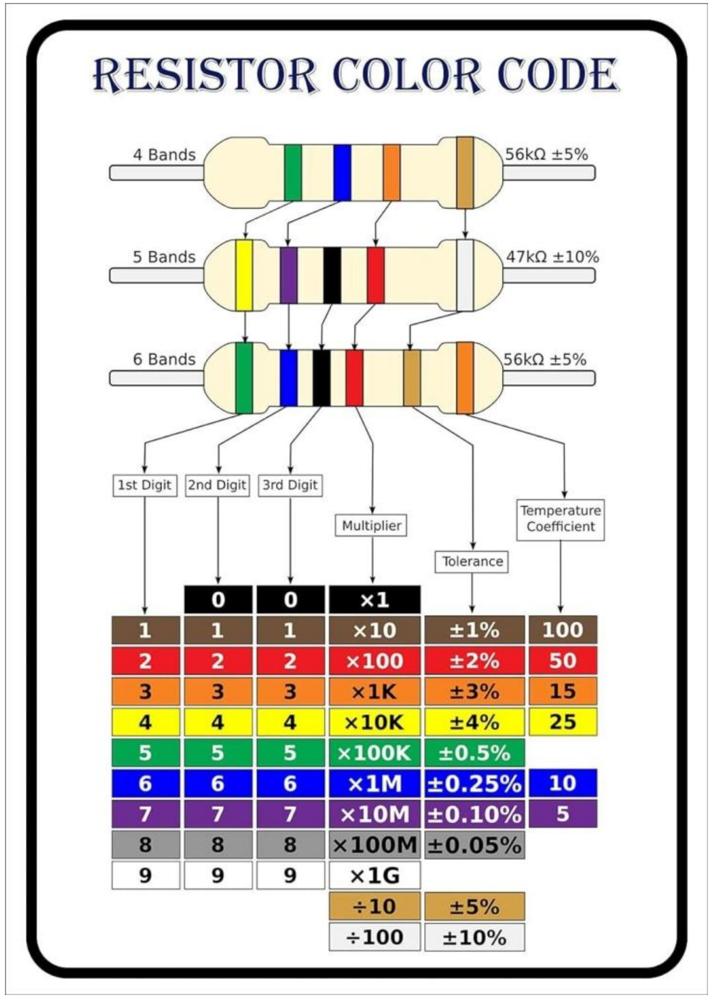


The positive side of the LED is called the "anode" and is marked by having a longer "lead," or leg. The other, negative side of the LED is called the "cathode."

Champlain
COLLEGE-LENNOXVILLE

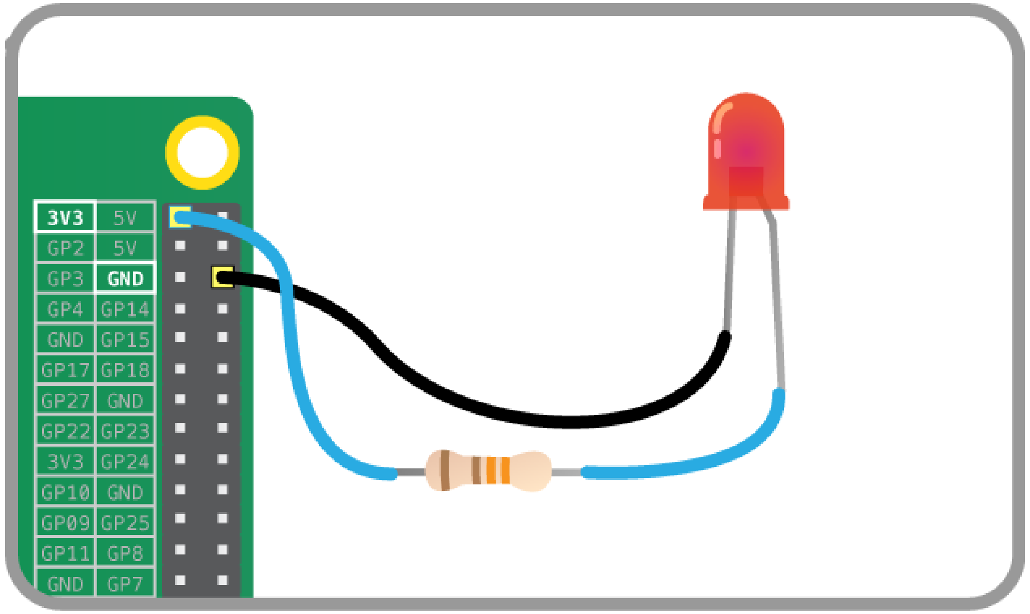
What is the resistor

- The main function of the resistor is to limit current. In the circuit, the character ‘R’ represents resistor, and the unit of resistor is ohm(Ω)).
- The band resistor is used in this experiment. A band resistor is one whose surface is coated with some particular color through which the resistance can be identified directly.

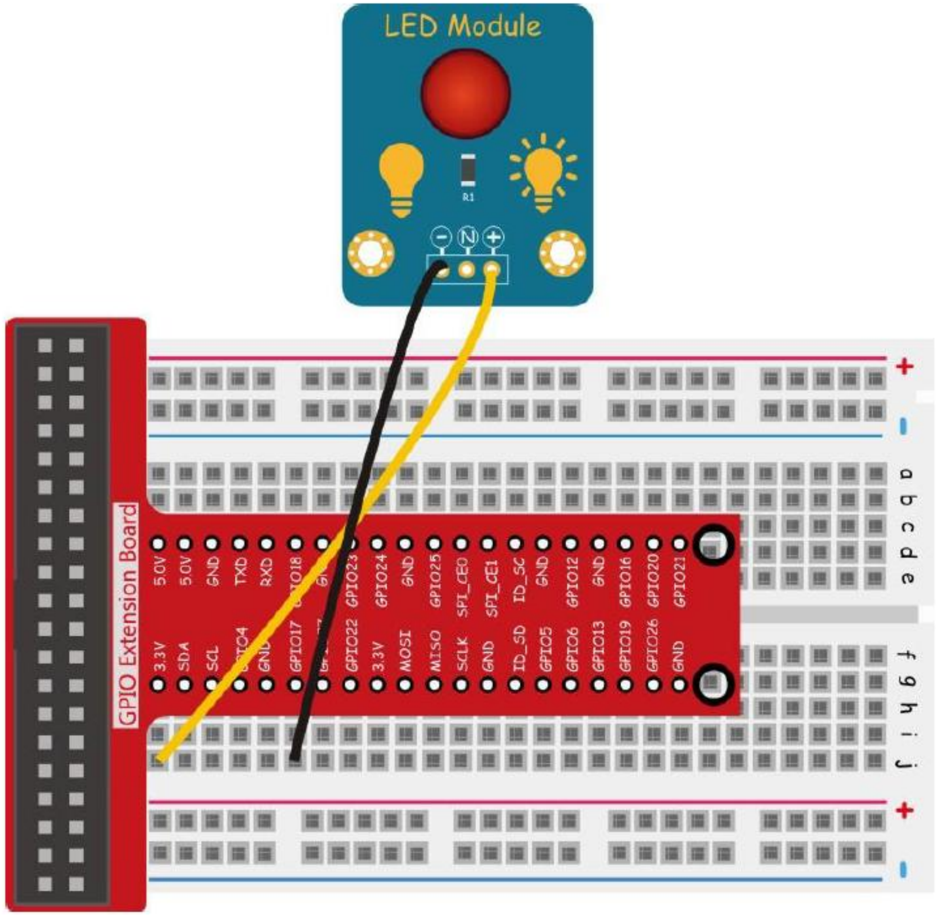


Lighting a LED

- To limit the current going through the LED, you should always use a resistor in series with it.
- Try connecting the long leg of an LED to the Pi's 3V3 and the short leg to a GND pin. The resistor can be anything over about 50Ω.
- As we used a dev kit, this can be simplified…



Lighting a LED



Lighting a LED

```
#/usr/bin/python3
import RPi.GPIO as GPIO
import time
LED=17 # on this pin I will connect my LED
GPIO.setmode(GPIO.BCM) # I am using the BCM naming
GPIO.setup(LED, GPIO.OUT) # GPIO17 is an output
GPIO.setup(LED, GPIO.HIGH) # LED starts OFF
blinks = 0 # initialize the blink
print ('Blinking starts')
while (blinks < 10): #let's do this 10 times
   GPIO.setup(LED, GPIO.LOW)
   print('LED ON')
   time.sleep(1)
   GPIO.setup(LED, GPIO.LOW)
   print('LED OFF')
   time.sleep(1)
   blinks=blinks + 1
GPIO.cleanup() # close the door when you leave
print('done')
```

Pulse Width modulation

The fancy blinker

- You must write a lot of code to get the LED to blink, fortunately; the GPIO has a feature that can help to make that easier.
- PWM (Pulse width modulation) is a technique used in the digital world mainly to control the speed of motors using a pulsed digital signal.
- The more pulses per second, the faster the motor runs.
- You can apply this to your blinking project as well.

The fancy blinker

- With PWM, you control the amount of time the HIGH/LOW signals repeat (called the frequency) and the amount of time the signal stays in HIGH state (called the duty cycle)
- It just so happens that the Broadcom GPIO signal 18 doubles as a PWM signal.
- You can set the GPIO 18 signal to PWM mode by using GPIO.PWM() method.

```
blink = GPIO.PWM (channel, frequency)
```

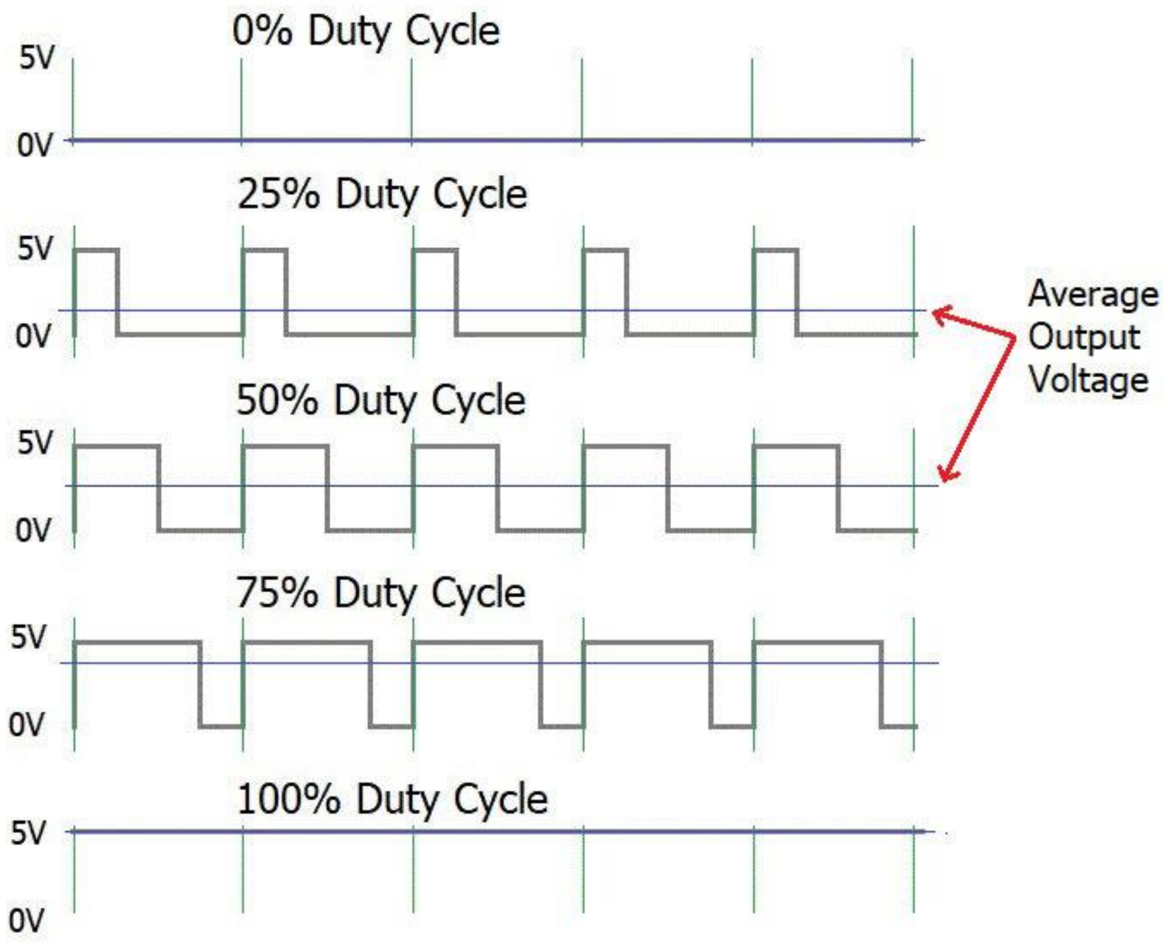
The fancy blinker

A word about frequency…

- Frequency is the number of occurrences of a repeating event per unit of time.
- Frequency is measured in hertz (Hz) which is equal to one event per second.
- `F=1 (one pulse per second)`
- `F=2 (two pulses per second)`
- `F=0.5 (one pulse every 2 seconds)`

The fancy blinker

- After you setup the GPIO 18 signal, you can start and stop it by using the `start()` and `stop()` methods.

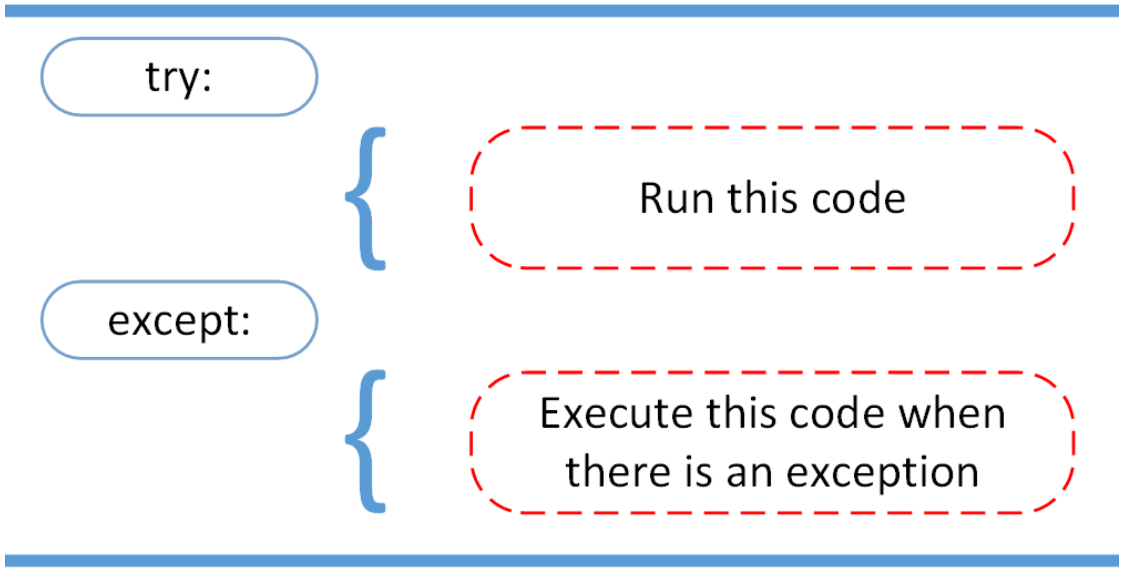


The fancy blinker

```python
#!/usr/bin/python3
# file name: blinkingLedPWM.py
import RPi.GPIO as GPIO
LED = 18
FREQUENCY = 1
DUTY = 50
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)
blink = GPIO.PWM(LED,FREQUENCY)
try:
  blink.start(DUTY)
  while True:
    pass
except KeyboardInterrupt:
  blink.stop()
finally:
  GPIO.cleanup()
```

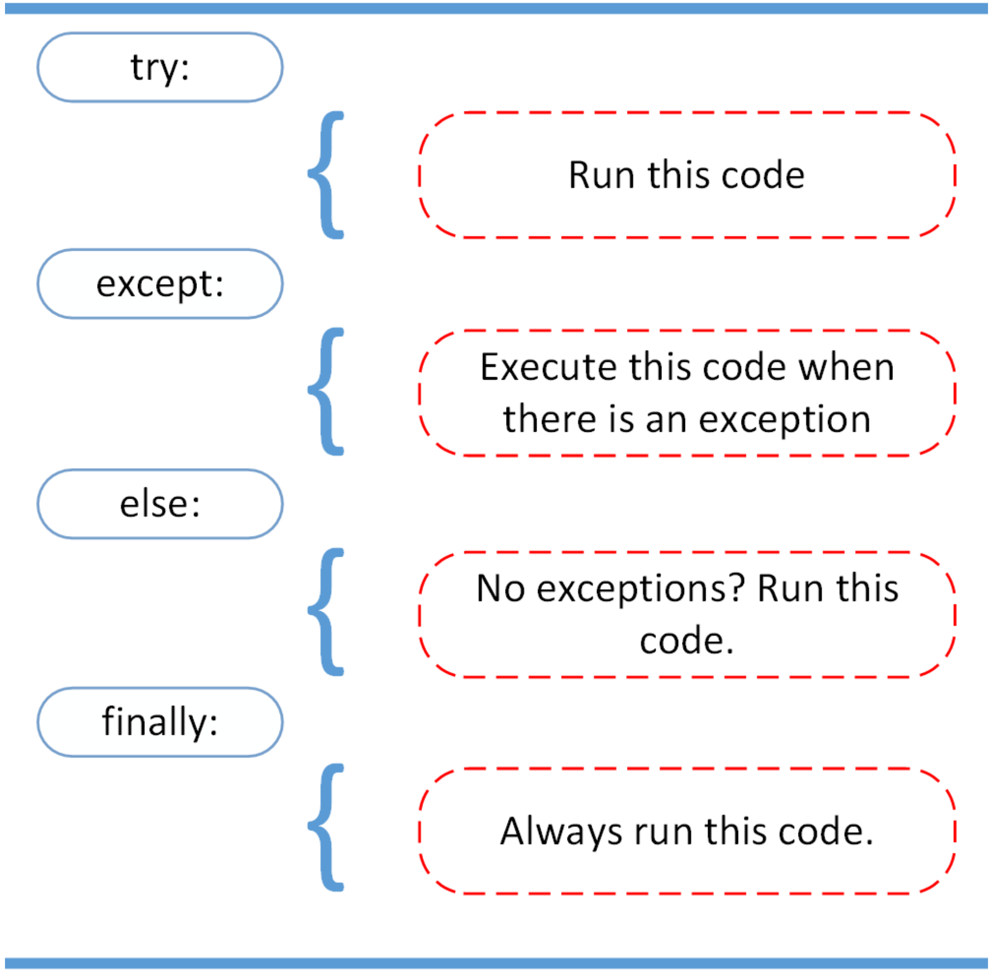
Handling Exceptions

- Python executes code following the try statement as a normal part of the program.
- The code that follows the except statement is the program's response to any exceptions in the preceding try clause:



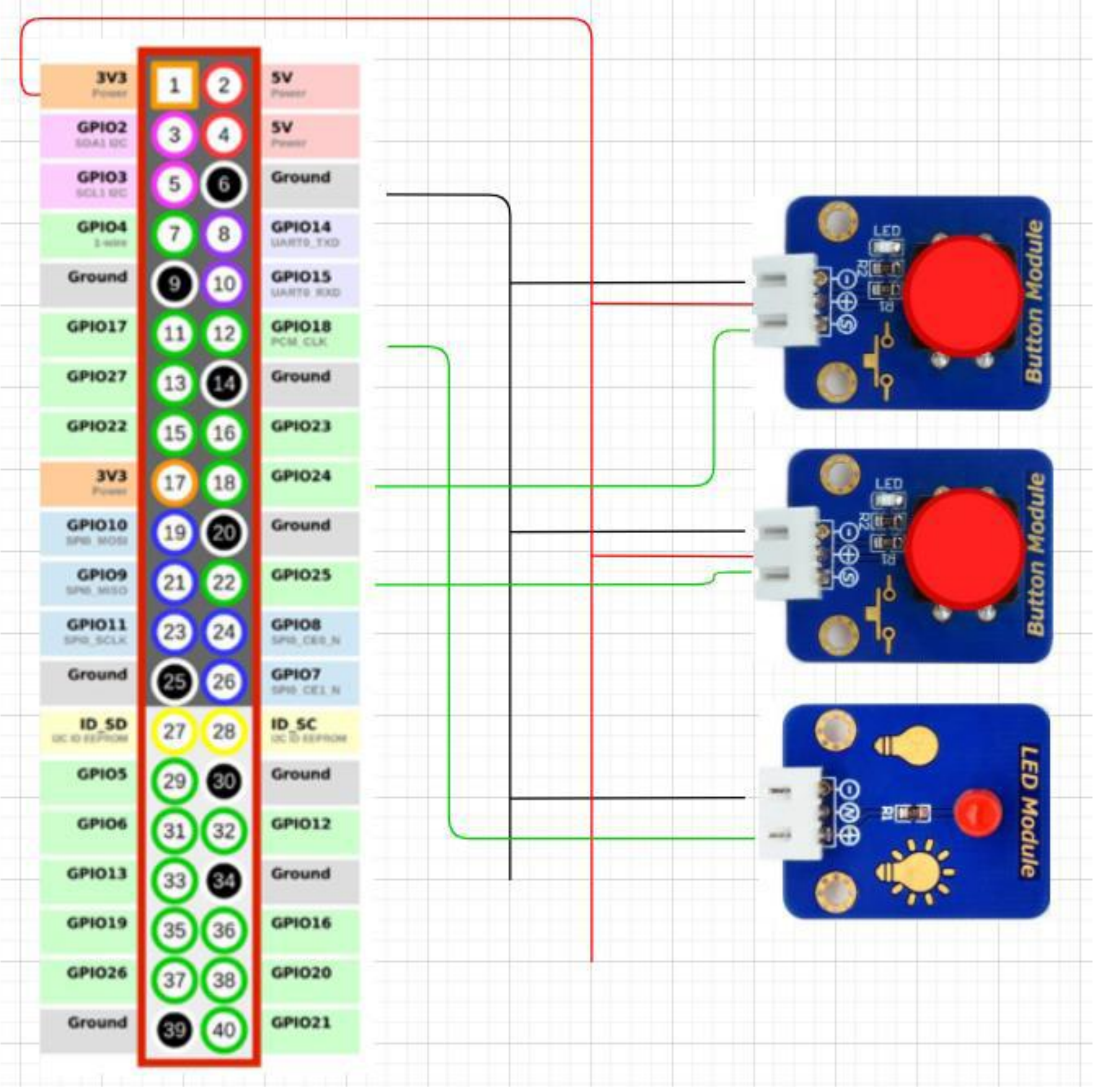
Cleaning Up After Execution

- You can use Python’s **else** statement to instruct a program to execute a certain block of code only in the absence of exceptions
- Imagine that you always had to implement some sort of action to clean up after executing your code. Python enables you to do so using the finally clause:



Detecting a GPIO input

The hardware setup



- We simulate a house with two doorbells and a light
  - One for the front door and one for the back door.
  - When someone rings one of the doorbells you will turn on the light.

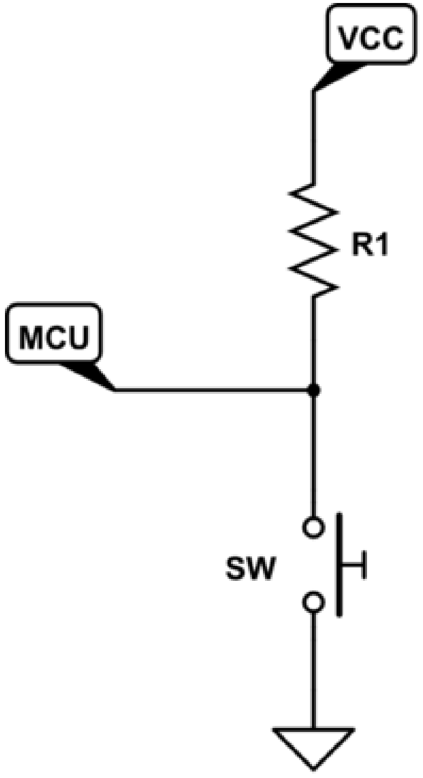
Detecting GPIO input

- Using the GPIO pins to detect input signals is a little bit trickier than using them for output.

RELASED

PRESSED

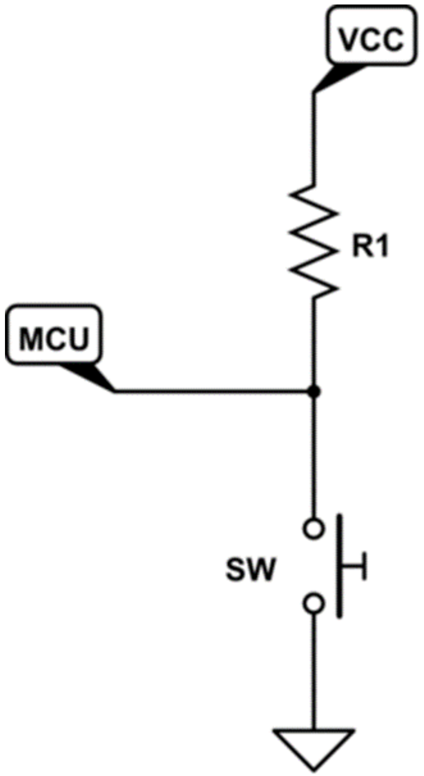
```
Shell
>>>
>>>
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(24, GPIO.IN)
>>> print (GPIO.input(24))
 1
>>> print (GPIO.input(24))
 0
>>>
```



Pull-up method

Detecting GPIO input

- A hidden problem exists whit this setup.
- Pushing the button connects the GPIO 24 pin to ground, forcing a LOW value .
- However, when the button isn't pressed, the GPIO 24 pin is not connected to anything.
- That means the pin could be in either HIGH or LOW state, and it may even switch back and forth without your doing anything.
- This is called ***flapping***



Pull-up method

Avoiding flapping

- To avoid flapping, you need to set the default value of the pin for when the button is not pressed.
- This is called a pull up (when you set the default value to HIGH) or pull down (when you set the default value to a LOW signal).

Avoiding flapping

- You can implement a pull up or pull down in this two ways:
- **Hardware .** Connect the GPIO 24 pin to either a 3.3 V voltage pin for a pull up (using a 10K to 50K ohm resistor to limit the current) or a GND pin (using a 1K ohm resistor) for a pull down.
- **Software.** The RPi.GPIO library provides the option of defining a pull up or pull down for the pin internally, using the option in the setup() method:

```
GPIO.setup(24, GPIO.IN, pull_up_down =GPIO.PUD_UP)
```

- Adding this line forces the GPIO 24 pin to always be in a HIGH status if the pin is not connected directly to ground.

Input Polling

- The most basic method used for reading a value from a switch is called polling
- The Python code checks the current value of a GPIO input pin at a regular interval.
- The GPIO input changing value means the switch was pressed.

Input Polling

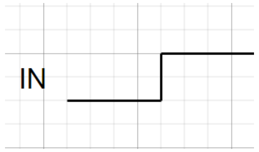
```python
#!/usr/bin/python3
# filename: TwoButtons1.py
import RPi.GPIO as GPIO
import time
LED=18
BUTTON1 = 24
BUTTON2 = 25

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)
GPIO.setup(BUTTON1, GPIO.IN, pull_up_down = GPIO.PUD_UP) #default UP
GPIO.setup(BUTTON2, GPIO.IN, pull_up_down = GPIO.PUD_UP) #default UP
try:
  while True:
    if (GPIO.input(BUTTON1) == GPIO.LOW):
       print ("Back door")
       GPIO.output(LED, GPIO.HIGH)   #LED ON
    elif (GPIO.input(BUTTON2) == GPIO.LOW):
       print ("Front door")
       GPIO.output(LED, GPIO.HIGH) #LED ON
    else:
       GPIO.output(LED, GPIO.LOW) #otherwise LED OFF
    time.sleep(0.1)
except KeyboardInterrupt:
  GPIO.cleanup()
print("End of the test")
```

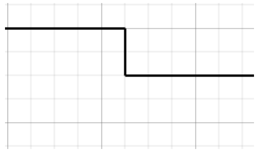
https://github.com/gabrielastudillo/Internet_of_Things_1/blob/main/week_10/TwoButtons1.py

Input Events

- For the polling, you must manually read the input value in each iteration and then determine whether the value has changed.
- Most of the time you are not interested in the value of the input at any specific moment. It is more interesting to detect when the ***value changes***.
  - Rising occurs when the input changes from LOW to HIGH, and



  - Falling happens when the input changes from HIGH to LOW.



Synchronous Events

- The wait_for_edge () method stops your program until it detects either a rising or falling event on the input signal.
- If you want your program to pause and wait for the event, this is the method to use

Synchronous Events

synchronousButton.py *

```python
#!/usr/bin/python3
import RPi.GPIO as GPIO

BUTTON1 = 24
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON1, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.wait_for_edge(BUTTON1, GPIO.FALLING)
print ('The Button 1 was pressed')
GPIO.cleanup()
```

Shell

```
>>> %Run synchronousButton.py
  The Button 1 was pressed
```

https://github.com/gabrielastudillo/Internet_of_Things_1/blob/main/week_10/synchronousButton.py

Asynchronous Events

- You don’t have to stop the entire program and wait for an event to occur. Instead, you can use asynchronous events.
- With asynchronous events, you can define multiple events for the program to listen for.
- Each event points to a method inside your code that runs when the event is triggered.
- You use the add_event_detect () method to define the event and the method to trigger:

```
GPIO.add_event_detect(channel, callback=method)
```

Asynchronous Events

asynchronusButton.py

```python
#/usr/bin/python3

import RPi.GPIO as GPIO
import time

LED = 18
BUTTON1 = 24
BUTTON2 = 25

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)
GPIO.output(LED, GPIO.LOW)
GPIO.setup(BUTTON1, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(BUTTON2, GPIO.IN, pull_up_down = GPIO.PUD_UP)

```

Asynchronous Events

asynchronusButton.py

```python
def backdoor(channel):
    GPIO.output(LED, GPIO.HIGH)
    print ('Back door')
    time.sleep(0.1)
    GPIO.output(LED, GPIO.LOW)

def frontdoor(channel):
    GPIO.output(LED, GPIO.HIGH)
    print ('Front door')
    time.sleep(0.1)
    GPIO.output(LED, GPIO.LOW)

GPIO.add_event_detect(BUTTON1, GPIO.FALLING, callback=backdoor)
GPIO.add_event_detect(BUTTON2, GPIO.FALLING, callback=frontdoor)
```

Asynchronous Events

asynchronusButton.py

```python
def backdoor(channel):
    GPIO.output(LED, GPIO.HIGH)
    print ('Back door')
    time.sleep(0.1)
    GPIO.output(LED, GPIO.LOW)

def frontdoor(channel):
    GPIO.output(LED, GPIO.HIGH)
    print ('Front door')
    time.sleep(0.1)
    GPIO.output(LED, GPIO.LOW)

GPIO.add_event_detect(BUTTON1, GPIO.FALLING, callback=backdoor)
GPIO.add_event_detect(BUTTON2, GPIO.FALLING, callback=frontdoor)
```

Asynchronous Events

asynchronusButton.py *

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

print ('End of program')
```

https://github.com/gabrielastudillo/Internet_of_Things_1/blob/main/week_10/asynchronousButton.py

Lab