

# **PYTHON-EMB**

## **La manipulation du GPIO**

Version 0.9



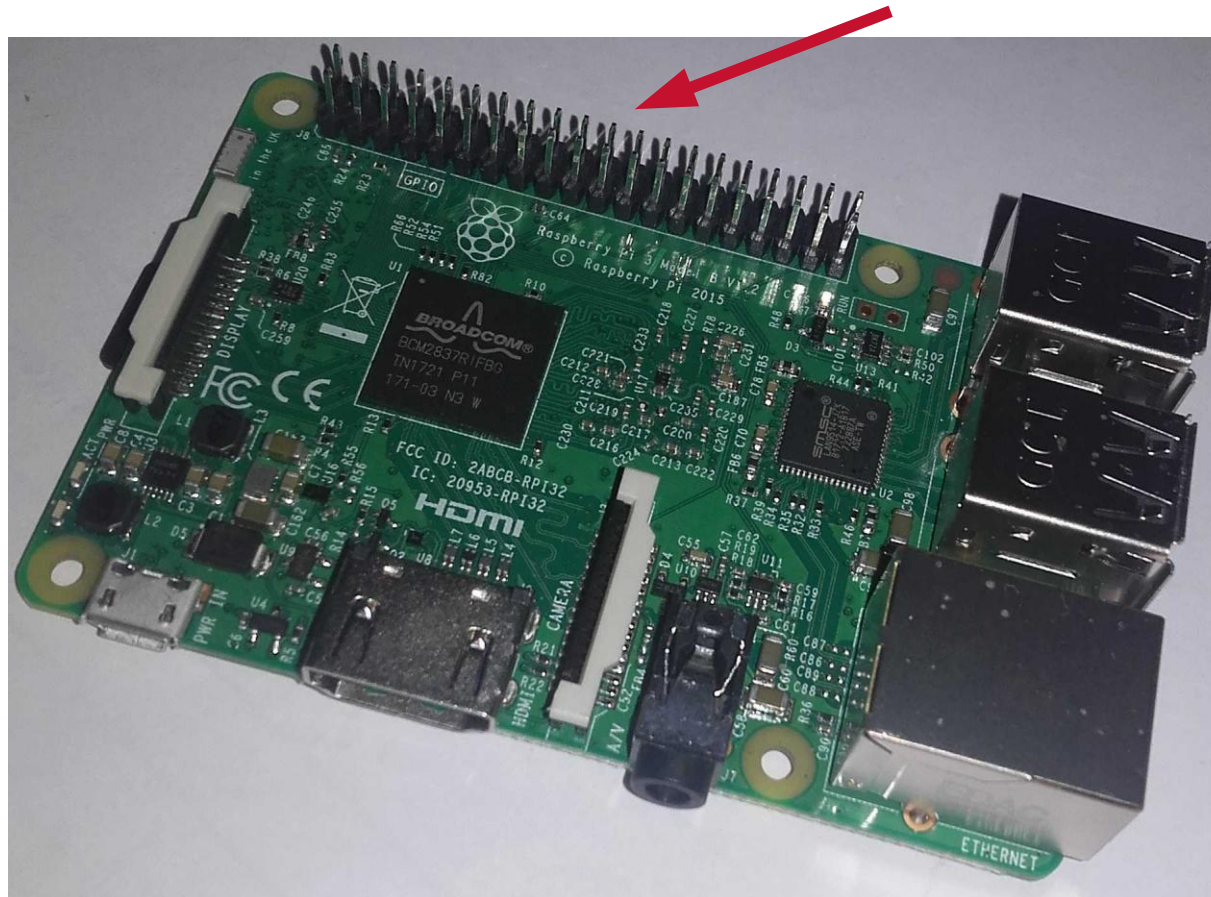
## Plan

- Le GPIO
- Prérequis pour la programmation en langage Python
- La configuration du GPIO
- La gestion des pins
- La gestion de l'état de sortie d'un pin pour allumer une led
- La lecture de l'état d'entrée d'un pin pour gérer un interrupteur
- Le PWM

**Le GPIO**

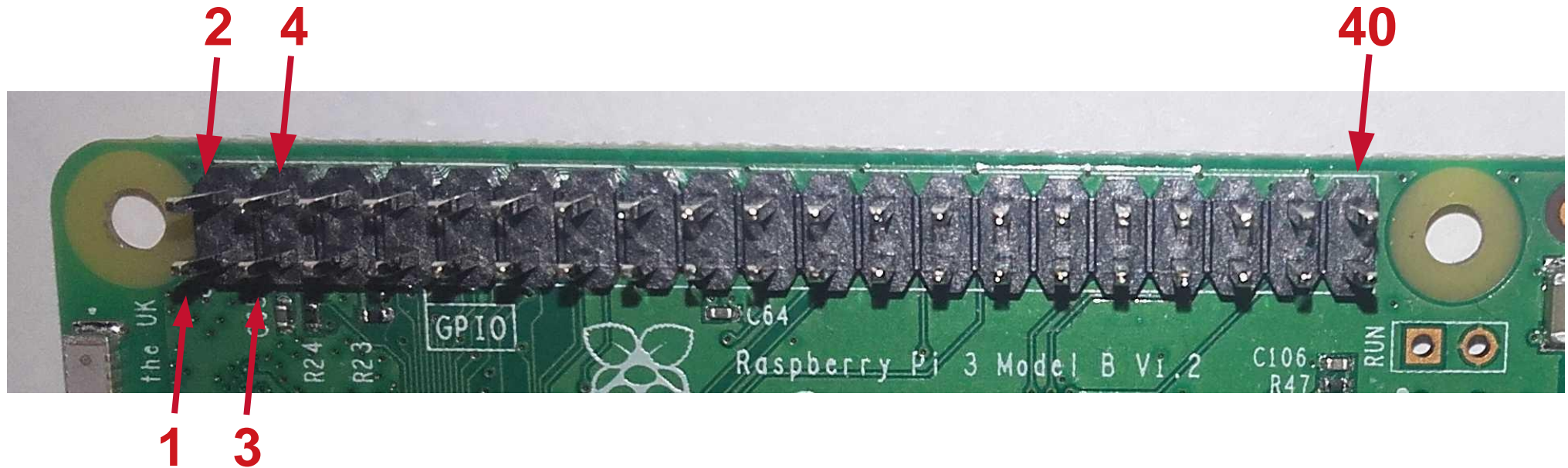
## 40 broches

Le Raspberry Pi 3 dispose d'un GPIO (General Purpose Input/Output) de 40 broches



## La numérotation des pins

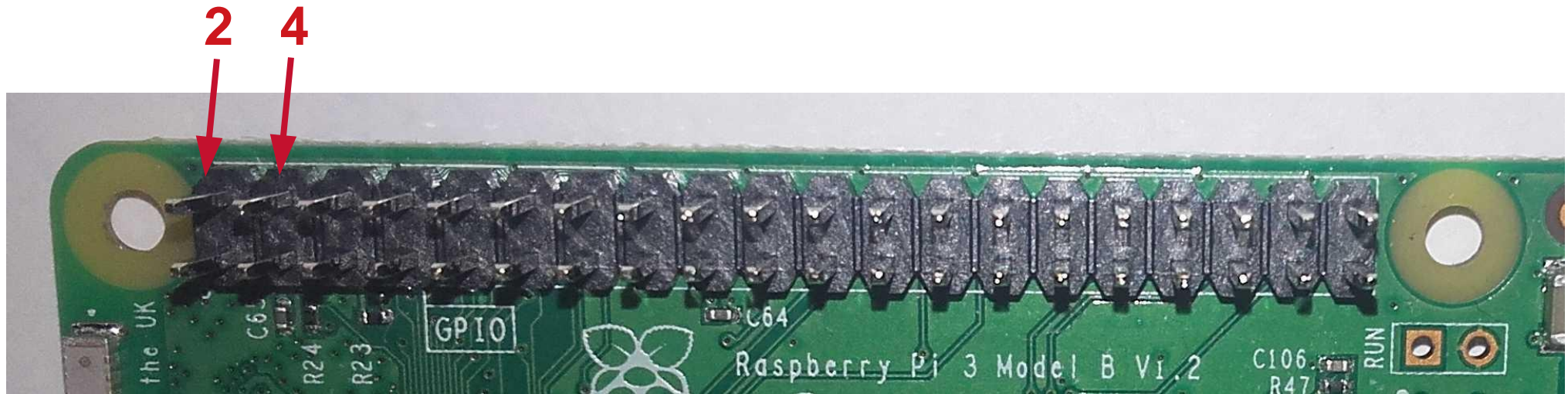
La pin n°1 se trouve en bas à gauche, la pin n°2 se trouve juste au dessus, la pin n°3 en 2<sup>ème</sup> position sur la rangée du bas... jusqu'à la pin n°40 en haut à droite.



## Les pins 5 v

Les pins 2 et 4 sont reliées au rail interne de 5 volt du Raspberry Pi.

Elles peuvent alimenter des dispositifs externes avec une intensité cumulée de 1,5 A si on utilise un bon transformateur pour alimenter le Raspberry Pi.





## Les pins 5 v

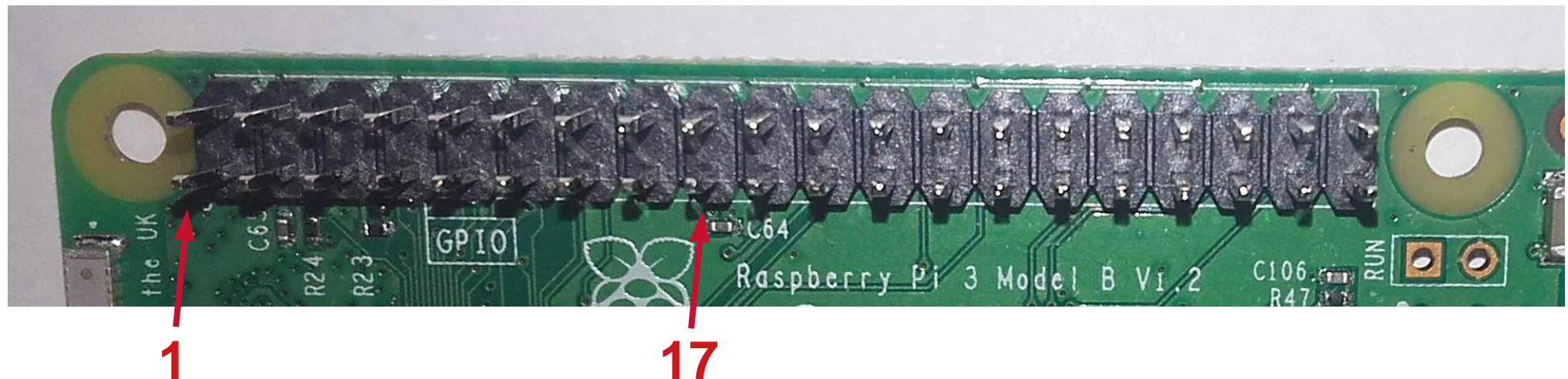
On peut également alimenter le Raspberry Pi par l'une de ces deux pins (il faut alors débrancher le transformateur) mais si l'intensité est insuffisante, le Raspberry Pi peut avoir un comportement erratique.



## Les pins 3,3 v

Les pins 1 et 17 sont reliées au rail interne de 3,3 volt du Raspberry Pi.

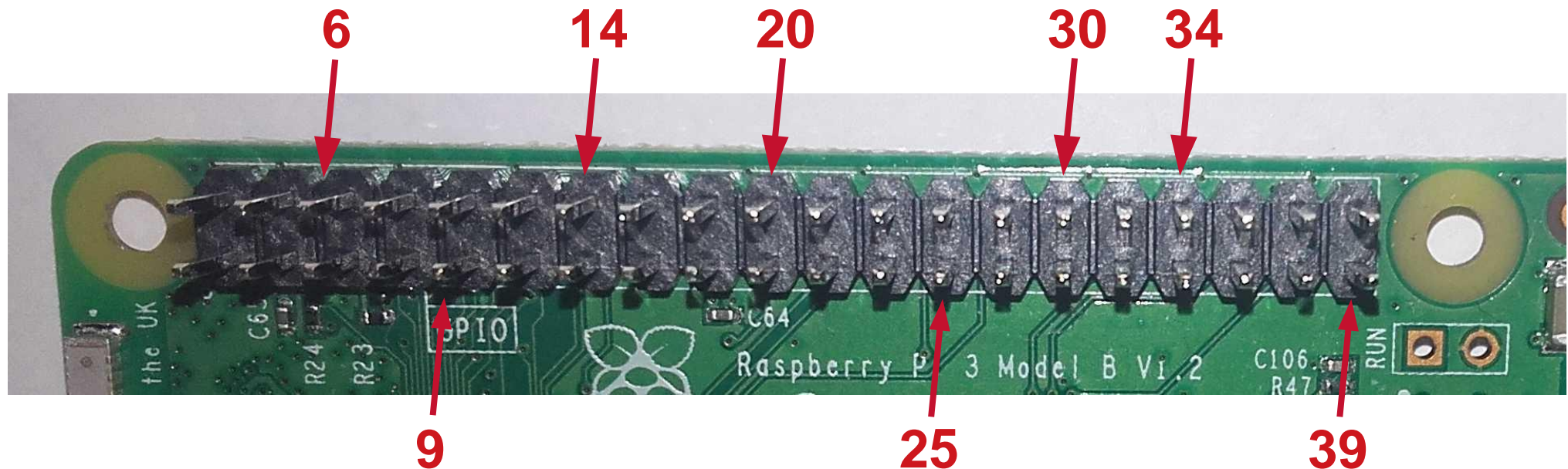
Elles peuvent alimenter des dispositifs externes avec une intensité cumulée de 500 mA.





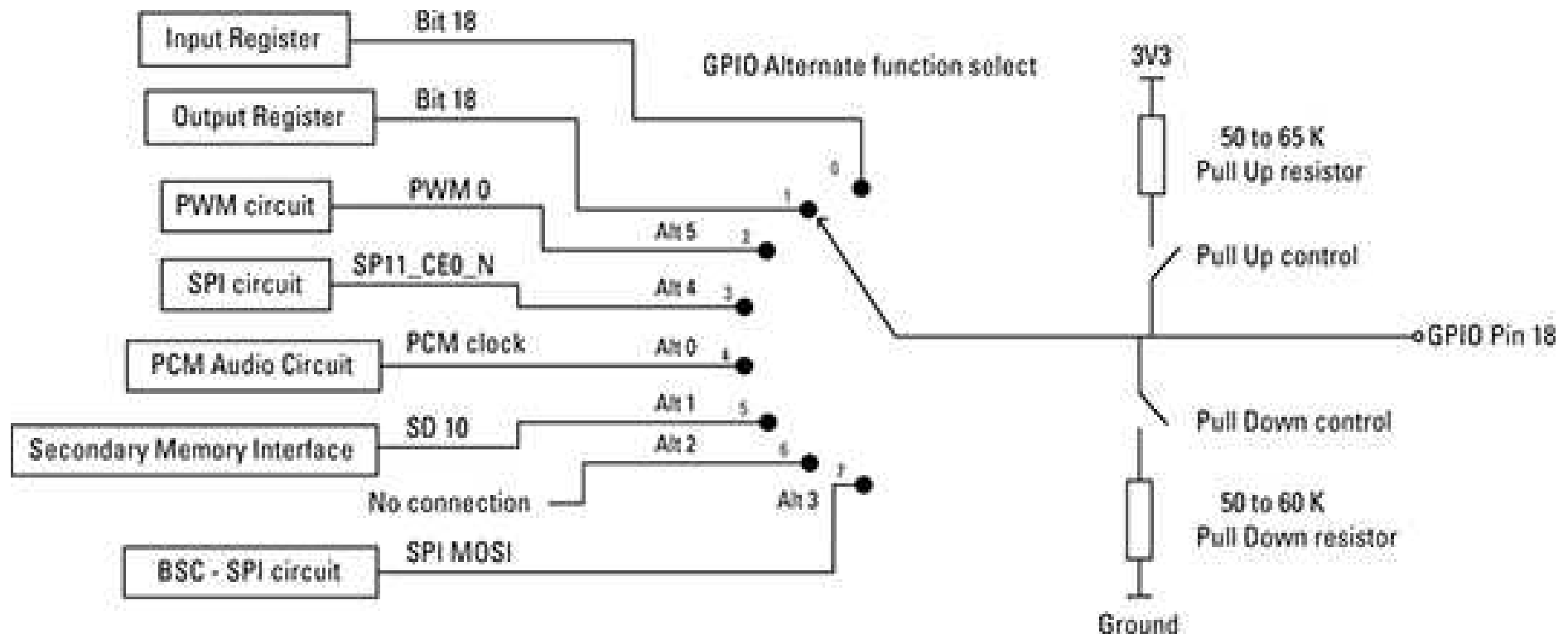
## Les pins « masse »

Les pins 6, 9, 14, 20, 25, 30, 34 et 39 sont reliées au rail interne de masse (ground) du Raspberry Pi qui est considéré comme faisant 0 volt (tension de référence).



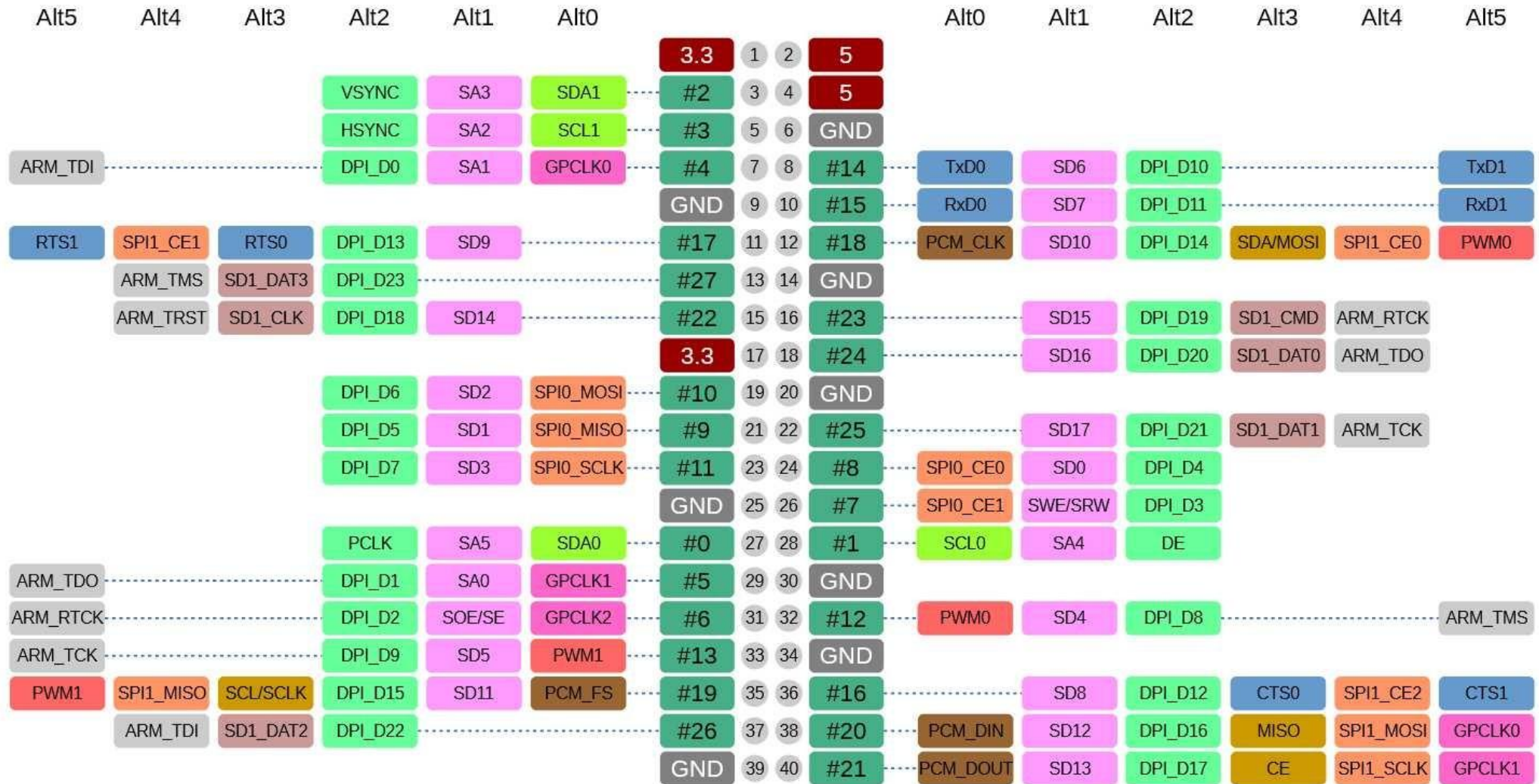
## Les différents modes de fonctionnement

Le fonctionnement des autres pins dépend en partie du mode de fonctionnement du Raspberry Pi.



Voir <http://www.dummies.com/computers/raspberry-pi/raspberry-pi-gpio-pin-alternate-functions/>

## Les différents modes de fonctionnement



Voir <https://franciscomoya.gitbooks.io/taller-de-raspberry-pi/content/es/elems/gpio.html>

## Les différents modes de fonctionnement

Les modes Input et Output sont gérés par Python alors que les modes Alt ne semblent pas accessibles depuis un script Python.

Pour accéder à ces autres modes, il faut manipuler les divers fichiers linux qui encapsulent le fonctionnement du GPIO ou utiliser une bibliothèque C telle que WiringPi.

En outre, certains pins d'entrées/sorties sont indisponibles selon les interfaces configurées dans le Raspberry Pi (voir la commande `raspi-config`)

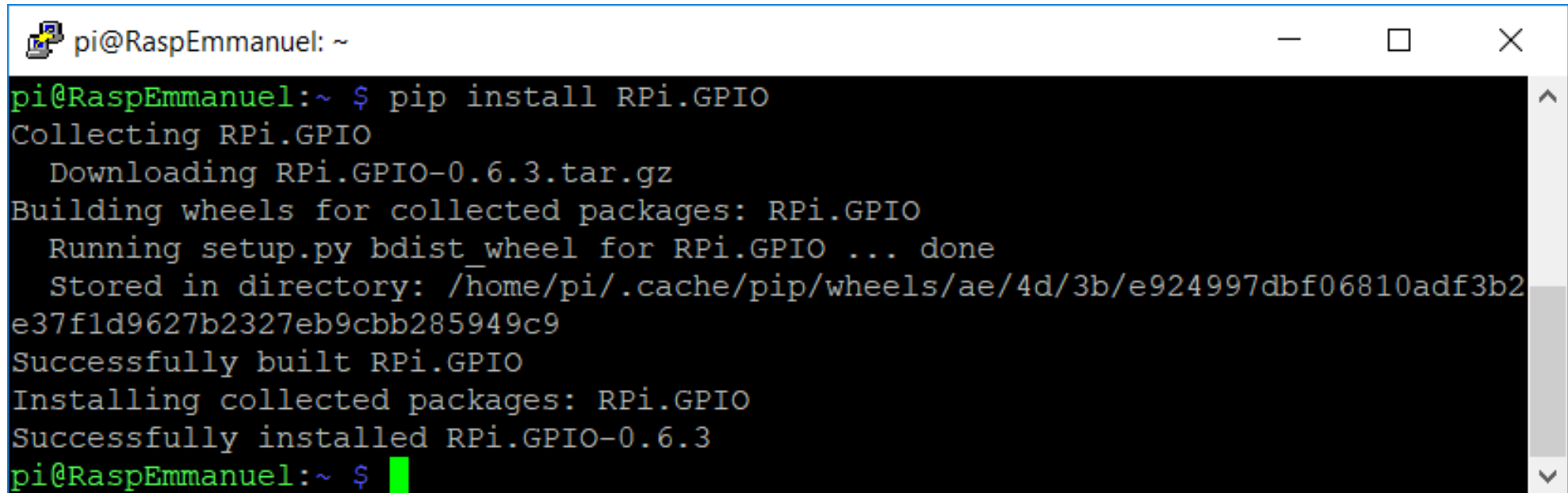
Voir <https://www.blaess.fr/christophe/2012/11/26/les-gpio-du-raspberry-pi/>

**Prérequis pour  
la  
programmation  
en langage  
Python**



# Installation

On installe d'abord le module « `RPi.GPIO` » en utilisant l'utilitaire `pip`.

A terminal window titled 'pi@RaspEmmanuel: ~' with standard window controls. The terminal output shows the command 'pip install RPi.GPIO' being executed. It details the collection of the package, downloading the tar.gz file, building the wheel, and finally installing it successfully. The prompt returns to the shell.

```
pi@RaspEmmanuel:~ $ pip install RPi.GPIO
Collecting RPi.GPIO
  Downloading RPi.GPIO-0.6.3.tar.gz
Building wheels for collected packages: RPi.GPIO
  Running setup.py bdist_wheel for RPi.GPIO ... done
  Stored in directory: /home/pi/.cache/pip/wheels/ae/4d/3b/e924997dbf06810adf3b2e37f1d9627b2327eb9cbb285949c9
Successfully built RPi.GPIO
Installing collected packages: RPi.GPIO
Successfully installed RPi.GPIO-0.6.3
pi@RaspEmmanuel:~ $
```

# Importation du module dans le code

On doit d'abord importer ce module GPIO afin que les instructions, qu'il contient, soient reconnues dans le script python.

```
import RPi.GPIO as GPIO
```

# **La configuration du GPIO**

# Sélection du mode de numérotation

Les pins du GPIO peuvent être désignées selon deux modes de numérotation :

- BOARD : la numérotation correspondant à celle du connecteur

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

- BCM : la numérotation utilisée par le microprocesseur (Broadcom SOC channel) et qu'on retrouve en partie sur le T-Cobbler

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

Voir <http://raspi.tv/2013/rpi-gpio-basics-4-setting-up-rpi-gpio-numbering-systems-and-inputs>  
<http://raspi.tv/2015/rpi-gpio-function-gpio-getmodev2>

# Déterminer le mode de numérotation

Le module `GPIO` propose une méthode `getmode()` qui peut renvoyer :

- « None » si aucun mode est configuré
- « 10 » si le mode « BOARD » est actif
- « 11 » si le mode « BCM » est actif



## Exemple de code

On considère le script ci-dessous.

```
#!/usr/bin/python3

from RPi import GPIO
from time import sleep

modes = {None:"Unset", 11:"BCM", 10:"BOARD"}

print ("Pi est {}".format (modes [ GPIO.getmode() ] ) )

sleep(3)

print (" => On change vers le mode BCM")
GPIO.setmode(GPIO.BCM)

print ("Pi est {}".format (modes [ GPIO.getmode() ] ) )

sleep(3)

GPIO.setup(24, GPIO.IN)      # A mettre sinon pas de changement de mode
GPIO.cleanup()              #
```

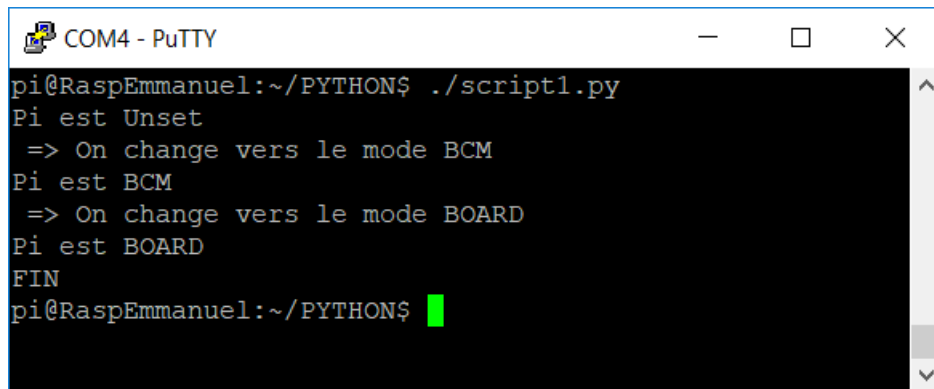
## Exemple de code

```
sleep(3)

print (" => On change vers le mode BOARD")
GPIO.setmode(GPIO.BOARD)

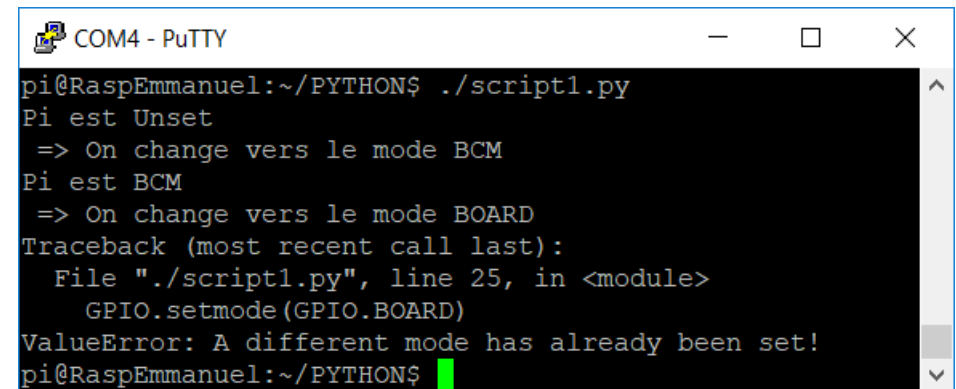
print ("Pi est {}".format (modes [ GPIO.getmode() ] ) )

print("FIN")
```



A screenshot of a PuTTY terminal window titled 'COM4 - PuTTY'. The terminal shows the execution of a Python script. The output is as follows:

```
pi@RaspEmmanuel:~/PYTHON$ ./script1.py
Pi est Unset
=> On change vers le mode BCM
Pi est BCM
=> On change vers le mode BOARD
Pi est BOARD
FIN
pi@RaspEmmanuel:~/PYTHON$
```



A screenshot of a PuTTY terminal window titled 'COM4 - PuTTY'. The terminal shows the execution of a Python script. The output is as follows:

```
pi@RaspEmmanuel:~/PYTHON$ ./script1.py
Pi est Unset
=> On change vers le mode BCM
Pi est BCM
=> On change vers le mode BOARD
Traceback (most recent call last):
  File "./script1.py", line 25, in <module>
    GPIO.setmode(GPIO.BOARD)
ValueError: A different mode has already been set!
pi@RaspEmmanuel:~/PYTHON$
```

Si on omet les lignes en bleu

Voir <http://raspi.tv/2015/rpi-gpio-function-gpio-getmodev2>

# **La gestion des pins**

# Le sens de circulation de l'information

Une pin peut fonctionner dans un seul sens à un instant donné :

- en entrée (`GPIO.IN`) pour que le Raspberry puisse lire l'état de la pin (`GPIO.HIGH` pour 3,3 v et `GPIO.LOW` pour 0 v) ;
- en sortie (`GPIO.OUT`) pour que le Raspberry positionne l'état de la pin à `GPIO.HIGH` ou `GPIO.LOW`.

# Spécification du sens de circulation

Pour spécifier le sens de circulation d'une pin, on utilise l'instruction `GPIO.setup` qui prend généralement en paramètre :

- le numéro de la pin (associé au mode de numérotation) ;
- le sens de circulation (`GPIO.IN` ou `GPIO.OUT`) ;
- éventuellement l'état initial de la pin (`GPIO.HIGH` ou `GPIO.LOW`).

Exemple :

```
GPIO.setup (17, GPIO.OUT, initial=GPIO.LOW)
```



# Connaître la configuration d'une pin

Pour connaître la configuration d'une pin, on utilise l'instruction `GPIO.gpio_function` qui prend le numéro de la pin en paramètre et renvoie une valeur qui peut être :

- `GPIO.INPUT ;`
- `GPIO.OUTPUT ;`
- `GPIO.SPI ;`
- `GPIO.I2C ;`
- `GPIO.HARD_PWM ;`
- `GPIO.SERIAL ;`
- `GPIO.UNKNOWN.`

# Connaître l'état d'une pin

Pour connaître l'état d'une pin, on utilise l'instruction `GPIO.input` qui prend le numéro de la pin en paramètre et renvoie une valeur qui peut être :

- `GPIO.HIGH` ;
- `GPIO.LOW`.

# Positionner l'état d'une pin

Pour positionner l'état d'une pin, on utilise l'instruction `GPIO.output` qui prend en paramètres :

- Le numéro de la pin concernée ;
- Le nouvel état de la pin qui peut être :
  - `GPIO.LOW` ;
  - `GPIO.HIGH` ;
  - `not GPIO.input (numéro)` pour inverser l'état de la pin sans nécessairement le connaître au sein du programme (toggle)

# Réinitialiser l'état d'une pin

A la fin d'un programme, il est conseillé de réinitialiser l'état des pins qui ont été modifiées. Pour positionner l'état d'une pin, on utilise l'instruction `GPIO.cleanup` qui ne prend aucun paramètre.

## Exemple 1

Le script ci-dessous permet de lister l'état de configuration d'une partie des 40 pins du GPIO.

```
#!/usr/bin/python

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

ports = [ 3, 5, 7, 8, 10, 11, 12, 13, 15, 16, 18, 19, 21,
          22, 23, 24, 26, 29, 31, 32, 33, 35, 36, 37, 38, 40 ]

port_use = { 0:"GPIO.OUT"      , 1:"GPIO.IN"      ,
             40:"GPIO.SERIAL"  , 41:"GPIO.SPI"   ,
             42:"GPIO.I2C"     , 43:"GPIO.HARD_PWM",
             -1:"GPIO.UNKNOWN" }

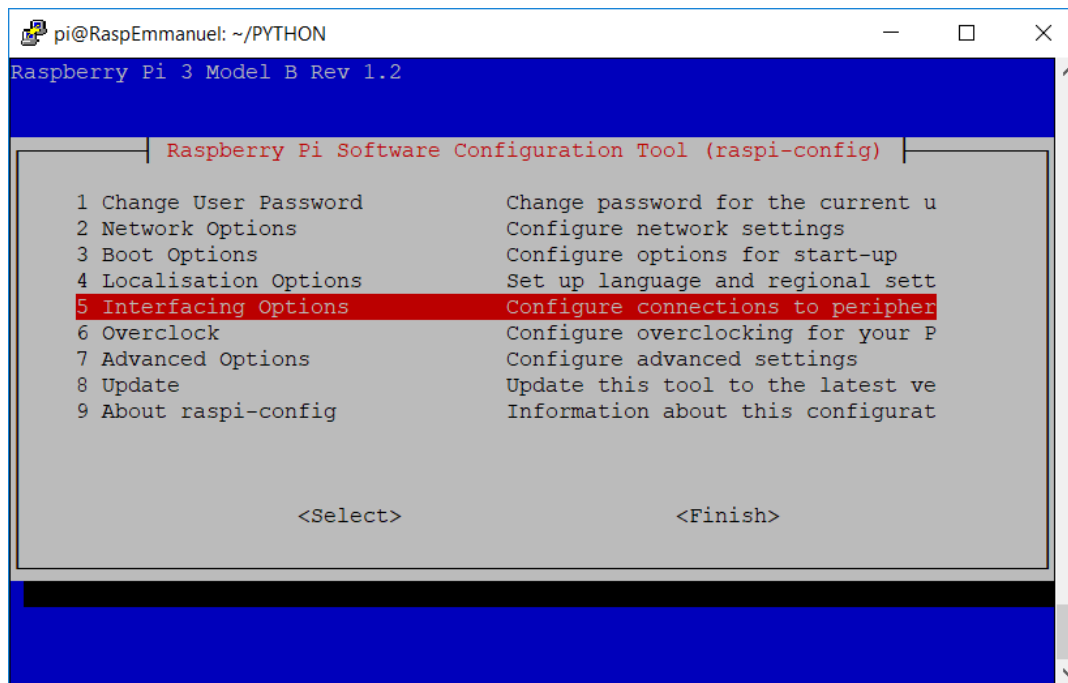
for p in ports :
    usage = GPIO.gpio_function (p)
    print "La pin %2d est dans le mode %s" % (p, port_use[usage])
```

Voir <http://raspi.tv/2014/rpi-gpio-port-function-checker>



## Exemple 1

Avant d'exécuter le script, on prend soin de désactiver toutes les interfaces, en utilisant « `sudo raspi-config` ».

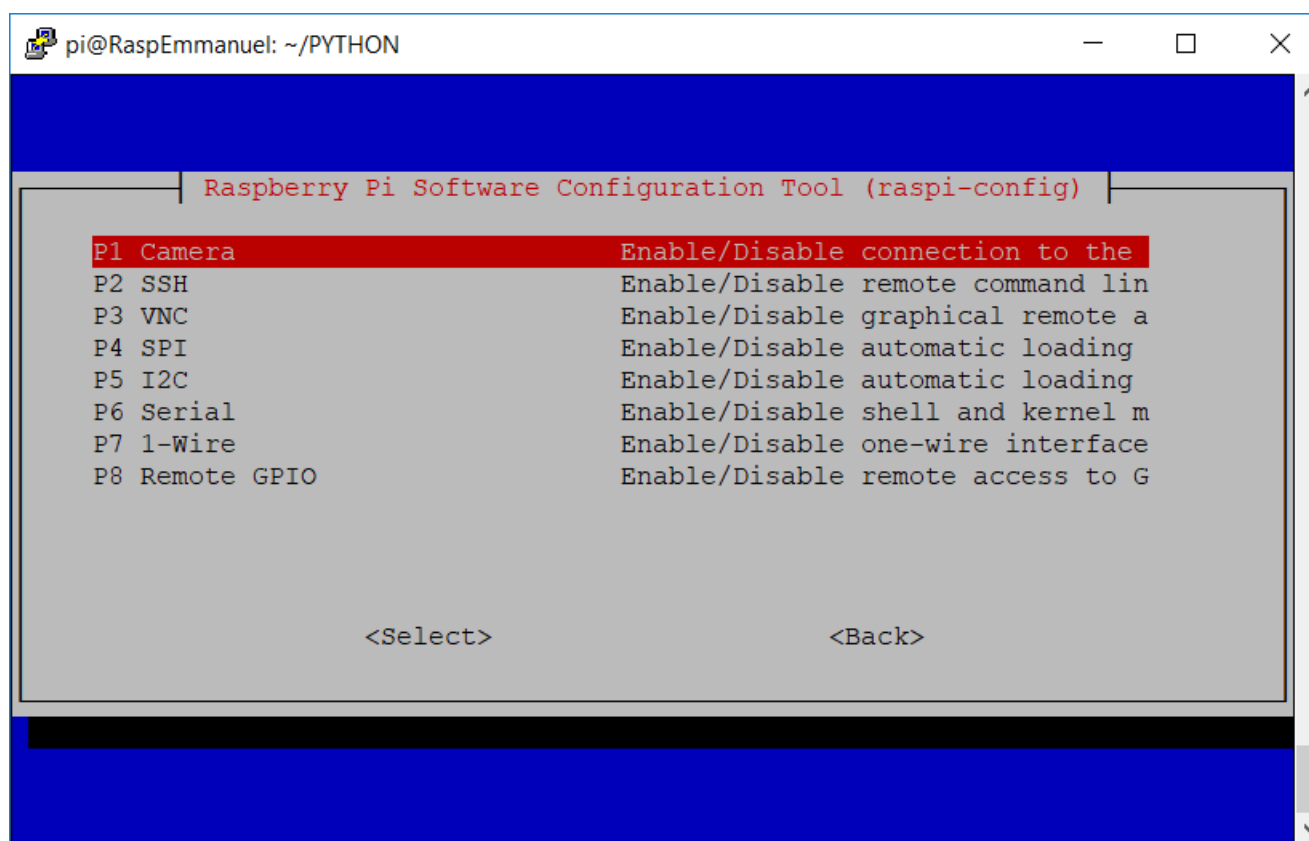


cette commande nous propose une interface en mode texte, dans laquelle on sélectionne l'item « Interfacing options ».

On appuie alors sur les touches [TAB] puis [ENTREE] pour se rendre dans le sous-menu.

## Exemple 1

A ce niveau, on peut désélectionner les interfaces SPI, I2C et Serial.

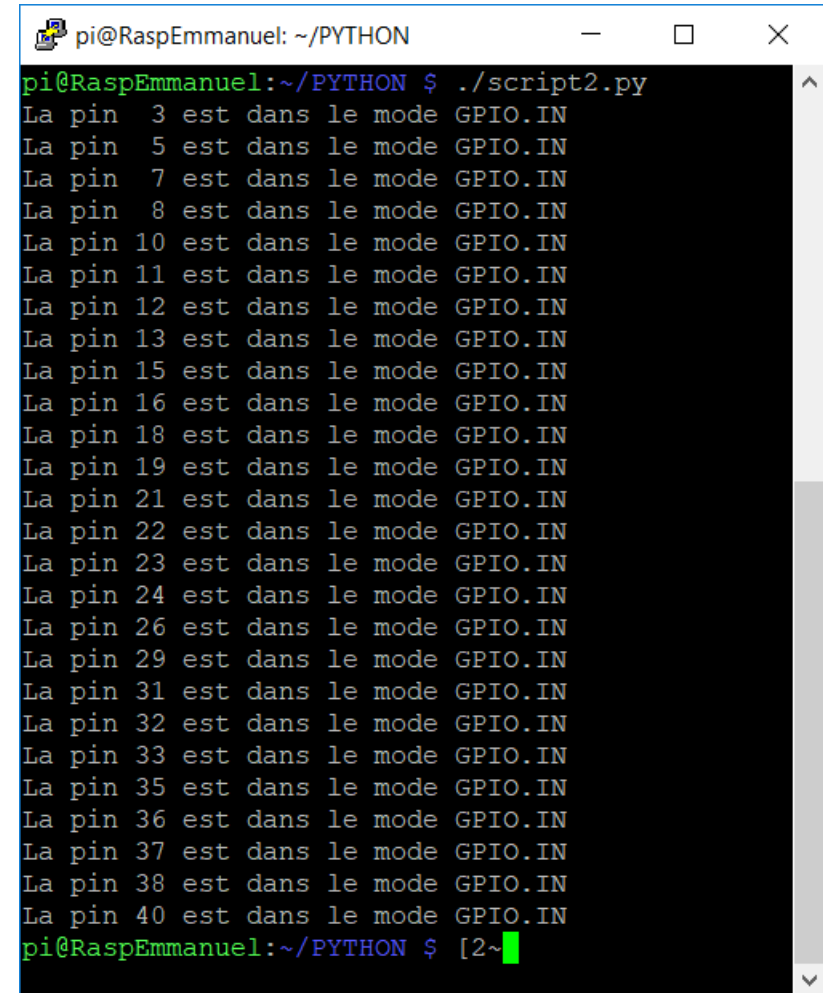


## Exemple 1

Une fois cette configuration effectuée, la commande nous propose de rebooter la Raspberry Pi.

A l'issue de ce redémarrage, on peut lancer le script afin d'obtenir le résultat ci-contre.

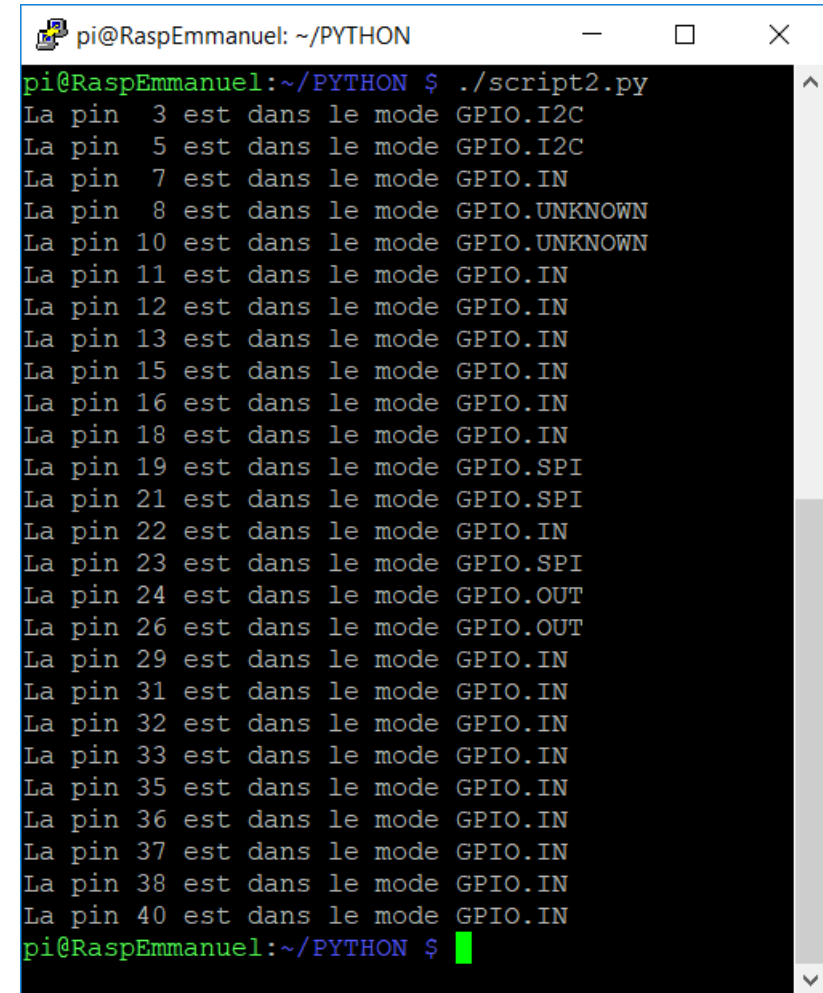
Cela correspond à la liste maximale des pins qui peuvent être utilisée en entrée/sortie « classique ».

A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' with standard window controls. The prompt is 'pi@RaspEmmanuel:~/PYTHON \$' and the command executed is './script2.py'. The output lists 20 pins (3, 5, 7, 8, 10, 11, 12, 13, 15, 16, 18, 19, 21, 22, 23, 24, 26, 29, 31, 32, 33, 35, 36, 37, 38, 40) and their mode 'GPIO.IN'. The prompt at the bottom is 'pi@RaspEmmanuel:~/PYTHON \$ [2~' with a green cursor.

```
pi@RaspEmmanuel:~/PYTHON $ ./script2.py
La pin 3 est dans le mode GPIO.IN
La pin 5 est dans le mode GPIO.IN
La pin 7 est dans le mode GPIO.IN
La pin 8 est dans le mode GPIO.IN
La pin 10 est dans le mode GPIO.IN
La pin 11 est dans le mode GPIO.IN
La pin 12 est dans le mode GPIO.IN
La pin 13 est dans le mode GPIO.IN
La pin 15 est dans le mode GPIO.IN
La pin 16 est dans le mode GPIO.IN
La pin 18 est dans le mode GPIO.IN
La pin 19 est dans le mode GPIO.IN
La pin 21 est dans le mode GPIO.IN
La pin 22 est dans le mode GPIO.IN
La pin 23 est dans le mode GPIO.IN
La pin 24 est dans le mode GPIO.IN
La pin 26 est dans le mode GPIO.IN
La pin 29 est dans le mode GPIO.IN
La pin 31 est dans le mode GPIO.IN
La pin 32 est dans le mode GPIO.IN
La pin 33 est dans le mode GPIO.IN
La pin 35 est dans le mode GPIO.IN
La pin 36 est dans le mode GPIO.IN
La pin 37 est dans le mode GPIO.IN
La pin 38 est dans le mode GPIO.IN
La pin 40 est dans le mode GPIO.IN
pi@RaspEmmanuel:~/PYTHON $ [2~
```

## Exemple 1

Si on active différentes interfaces (via la commande raspi-config) et qu'on relance ce même script, on obtient le résultat ci-contre.



```
pi@RaspEmmanuel: ~/PYTHON
pi@RaspEmmanuel:~/PYTHON $ ./script2.py
La pin 3 est dans le mode GPIO.I2C
La pin 5 est dans le mode GPIO.I2C
La pin 7 est dans le mode GPIO.IN
La pin 8 est dans le mode GPIO.UNKNOWN
La pin 10 est dans le mode GPIO.UNKNOWN
La pin 11 est dans le mode GPIO.IN
La pin 12 est dans le mode GPIO.IN
La pin 13 est dans le mode GPIO.IN
La pin 15 est dans le mode GPIO.IN
La pin 16 est dans le mode GPIO.IN
La pin 18 est dans le mode GPIO.IN
La pin 19 est dans le mode GPIO.SPI
La pin 21 est dans le mode GPIO.SPI
La pin 22 est dans le mode GPIO.IN
La pin 23 est dans le mode GPIO.SPI
La pin 24 est dans le mode GPIO.OUT
La pin 26 est dans le mode GPIO.OUT
La pin 29 est dans le mode GPIO.IN
La pin 31 est dans le mode GPIO.IN
La pin 32 est dans le mode GPIO.IN
La pin 33 est dans le mode GPIO.IN
La pin 35 est dans le mode GPIO.IN
La pin 36 est dans le mode GPIO.IN
La pin 37 est dans le mode GPIO.IN
La pin 38 est dans le mode GPIO.IN
La pin 40 est dans le mode GPIO.IN
pi@RaspEmmanuel:~/PYTHON $
```

**La gestion de  
l'état de sortie  
d'un pin pour  
allumer une led**

# Le script python à exécuter

L'exemple ci-dessous permet d'allumer une diode branché sur la pin BCM #21.

```
#!/usr/bin/python3

import RPi.GPIO as GPIO

GPIO.setmode (GPIO.BCM)

GPIO.setup (21, GPIO.OUT, initial=GPIO.HIGH)

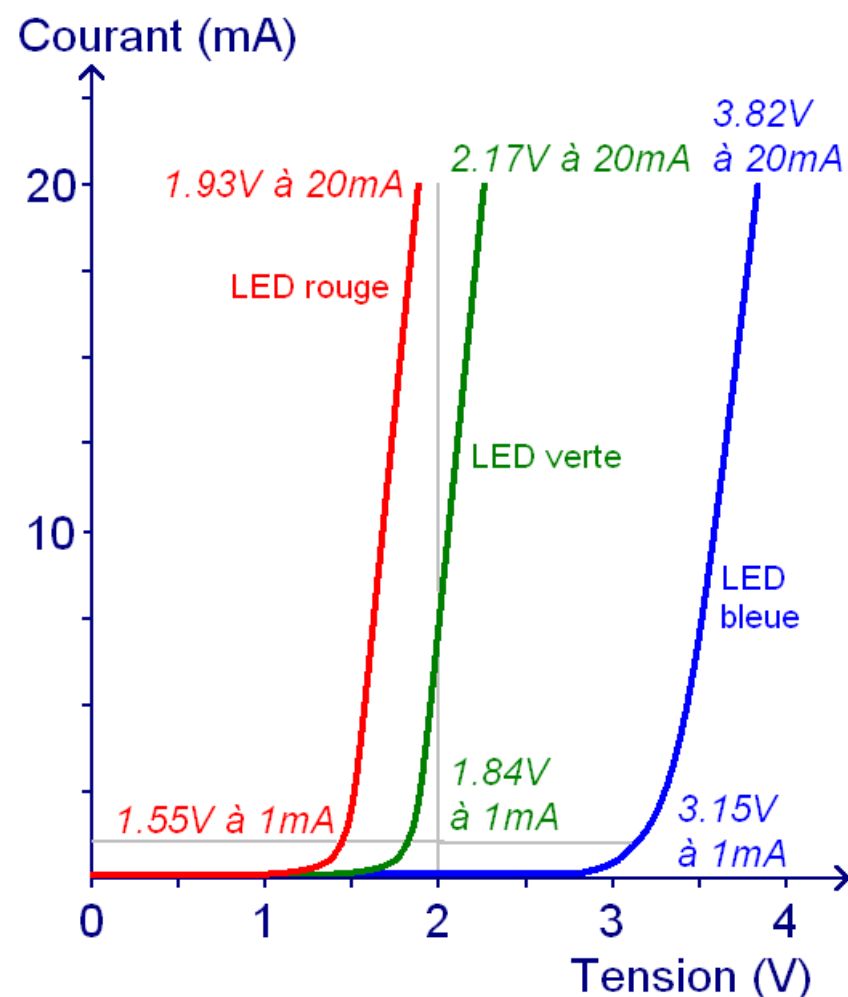
touche = input ('Frappez une touche')

GPIO.output (21, not GPIO.input (21))

GPIO.cleanup ()
```

# Comportement électrique d'une led

D'un point de vue électronique, cette pin délivre une tension de 3,3 v avec un courant de 500 mA. Il est donc nécessaire de prendre des précautions sous peine de griller la led qui ne supporte généralement qu'une intensité de 20 mA pour une tension variant de 1,93 v (rouge) à 3,82 v (bleu).



Voir <https://www.astuces-pratiques.fr/electronique/led-et-calcul-de-la-resistance-serie>

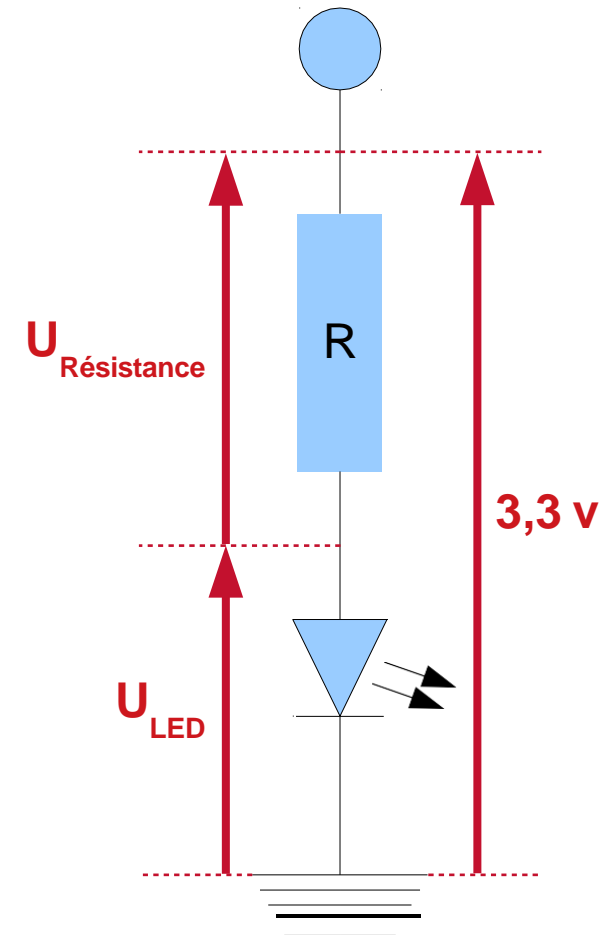
## Ajout d'une résistance anti-claquage

Cette précaution prend la forme d'une résistance placée entre la pin et la led afin de faire chuter la tension et l'intensité.

Pour calculer la valeur de cette résistance, on considère que la somme

$$U_{\text{LED}} + U_{\text{Résistance}} = 3,3 \text{ volt}$$

On fixe  $U_{\text{LED}}$  à 1,8 volt,  $U_{\text{Résistance}}$  est donc égal à  $3,3 - 1,8$  soit 1,5 volt.

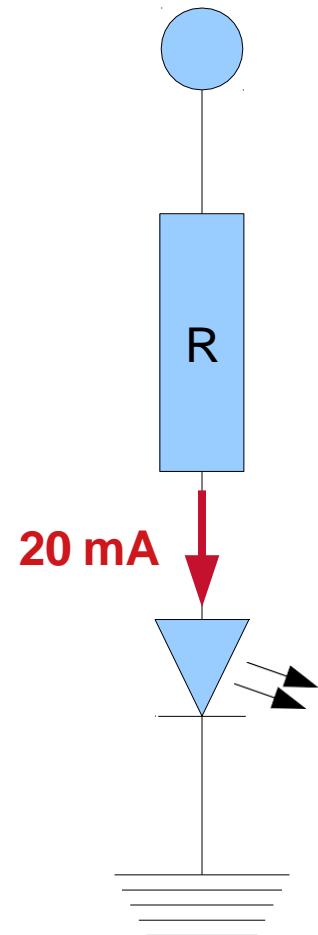




# Détermination de la valeur de la résistance

Comme on souhaite faire circuler un courant d'une intensité d'au plus 20 mA, on utilise la loi  $U=RI$  qui caractérise la résistance pour déterminer sa valeur  $R$ .

$$R = U/I = 3,3 / 0,02 = 165 \, \Omega$$



# Le code couleur des résistances

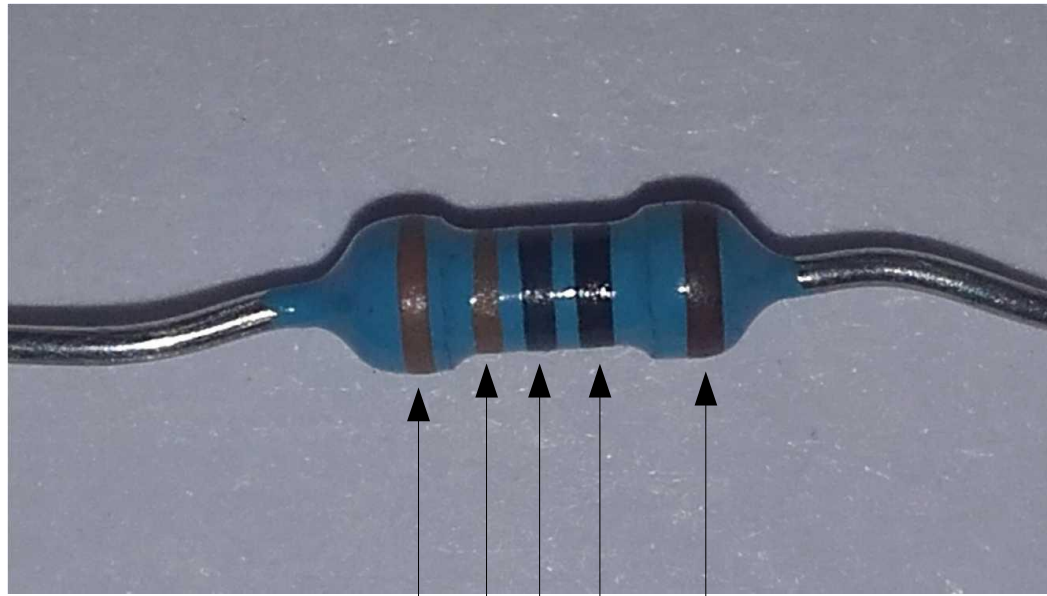
On doit rechercher une résistance d'au moins 165 ohms, il est alors nécessaire de connaître le code couleur de ces résistances pour en choisir une adaptée.

<div><div><div>0 1 2 3 4 5 6 7 8 9</div><div>0 Black</div><div>1 Brown</div><div>2 Red</div><div>3 Orange</div><div>4 Yellow</div><div>5 Green</div><div>6 Blue</div><div>7 Purple</div><div>8 Grey</div><div>9 White</div><div>±1% Brown</div><div>±2% Red</div><div>±5% Gold</div><div>±10% Silver</div></div><div><div>0 1 2 3 4 5 6 7 8 9</div><div>0 Black</div><div>1 Brown</div><div>2 Red</div><div>3 Orange</div><div>4 Yellow</div><div>5 Green</div><div>6 Blue</div><div>7 Purple</div><div>8 Grey</div><div>9 White</div><div>±1% Brown</div><div>±2% Red</div><div>±5% Gold</div><div>±10% Silver</div></div></div>	<div><div>±1%</div><div>±2%</div><div>±5%</div><div>±10%</div></div> <div><div>27K</div><div>EXAMPLE</div></div> <div><div>0 X1</div><div>1 1 X10</div><div>2 2 X100</div><div>3 3 X1000</div><div>4 4 X10000</div><div>5 5 X100000</div><div>6 6 X1000000</div><div>7 7 ÷10</div><div>8 8 ÷100</div><div>9 9</div></div> <div>Color Codes</div>	<div><div>±1%</div><div>±2%</div><div>±5%</div><div>±10%</div></div> <div><div>15K</div><div>EXAMPLE</div></div> <div><div>0 0 X1</div><div>1 1 1 X10</div><div>2 2 2 X100</div><div>3 3 3 X1000</div><div>4 4 4 X10000</div><div>5 5 5 ÷10</div><div>6 6 6 ÷100</div><div>7 7 7</div><div>8 8 8</div><div>9 9 9</div></div> <div>5 Band Resistors</div>	<div><div>±1%</div><div>±2%</div><div>±5%</div><div>±10%</div><div>100</div><div>50</div><div>25</div><div>15</div><div>10</div><div>5</div><div>1</div></div> <div><div>620K</div><div>EXAMPLE</div></div> <div><div>0 0 X1</div><div>1 1 1 X10</div><div>2 2 2 X100</div><div>3 3 3 X1000</div><div>4 4 4 X10000</div><div>5 5 5 ÷10</div><div>6 6 6 ÷100</div><div>7 7 7</div><div>8 8 8</div><div>9 9 9</div></div> <div>6 Band Resistors</div>
---	--	--	---

Voir <https://jkdgreat.wordpress.com/2012/10/16/resistor-color-code-identification-sheet/>  
<https://openclassrooms.com/courses/l-electronique-de-zero/resistance-et-resistor>

## Exemple de lecture d'une résistance

On considère, par exemple, la résistance ci-dessous.



Si on se réfère au code couleur (avec de bons yeux), on constate que la résistance a une valeur de  $330\ \Omega$ .

## Utilisation d'un ohmmètre

On peut vérifier cela avec un Ohmmètre (qui est bien plus facile à utiliser que le code couleur).

L'ohmmètre est réglé sur  $2k\Omega$ , il affiche donc des valeurs exprimées en  $k\Omega$ , jusqu'à  $2k\Omega$ .

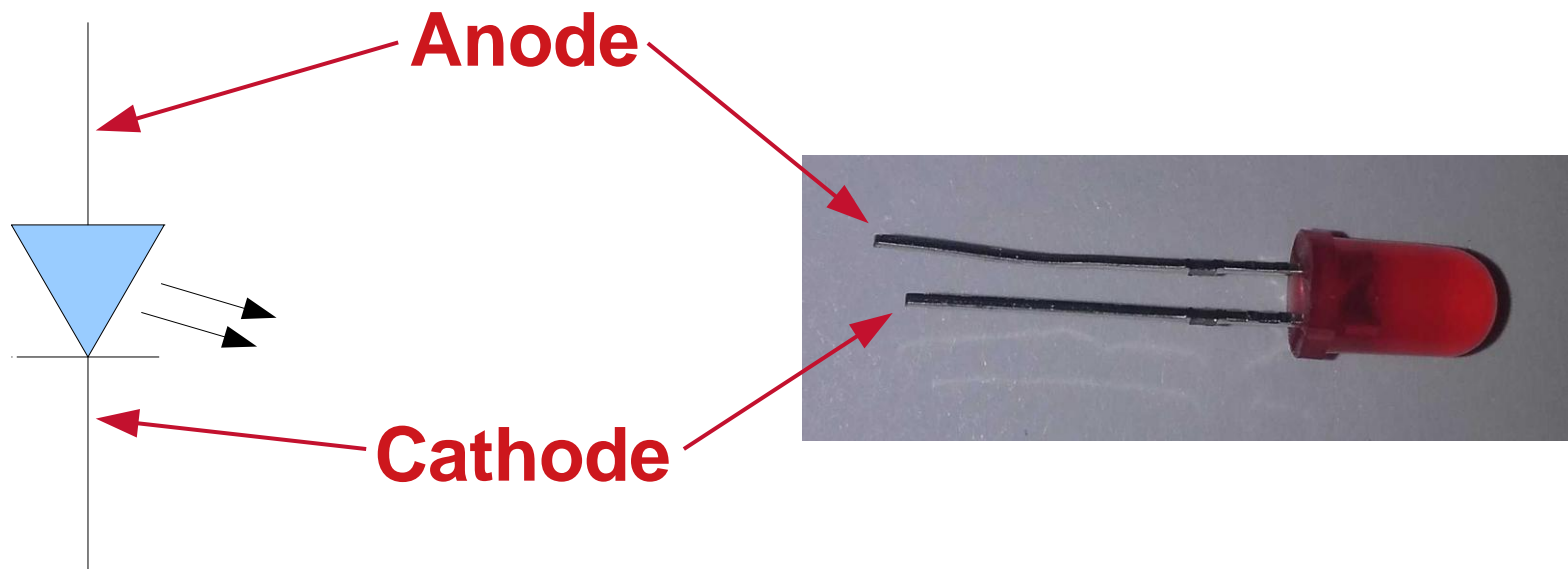
L'affichage 0,329 correspond à  $0,329k\Omega \approx 330\Omega$ .



## La led

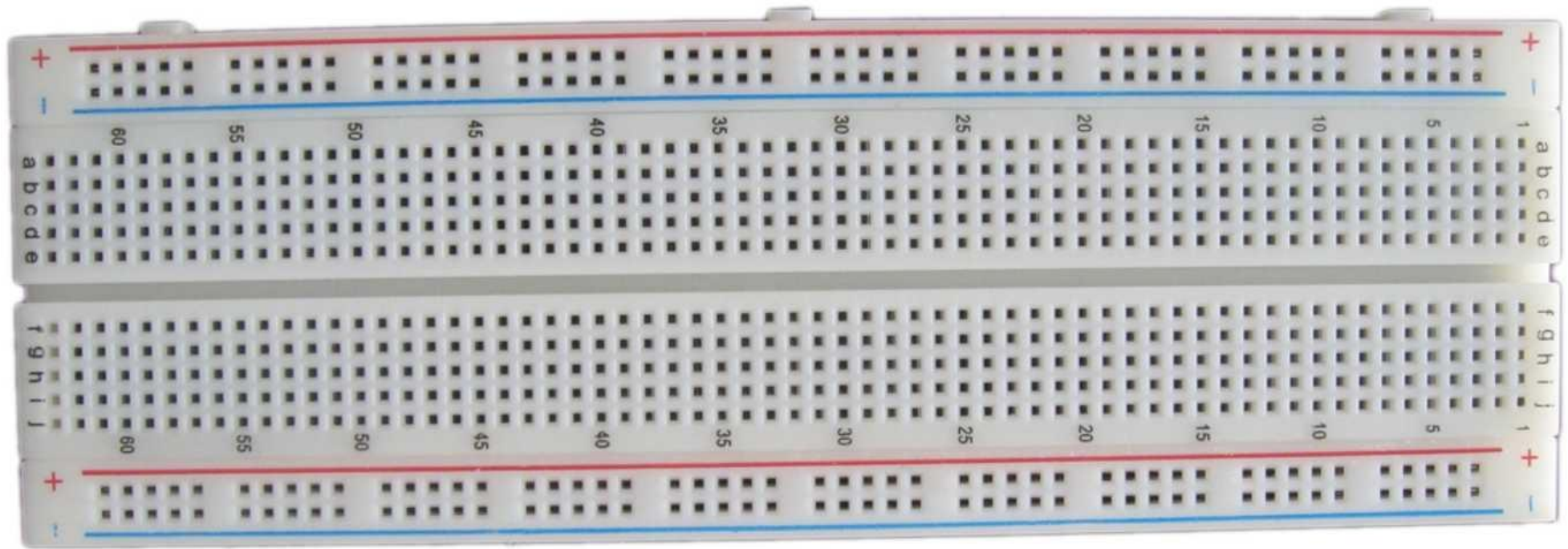
En ce qui concerne la diode, il faut la brancher dans le bon sens sous peine de la faire fonctionner en mode bloqué (et donc de ne pas la voir s'allumer) :

- l'anode sur le « plus » donc vers la sortie BCM #21 ;
- la cathode sur le « moins » donc vers la masse GND.



# La planche d'expérimentation sans soudure

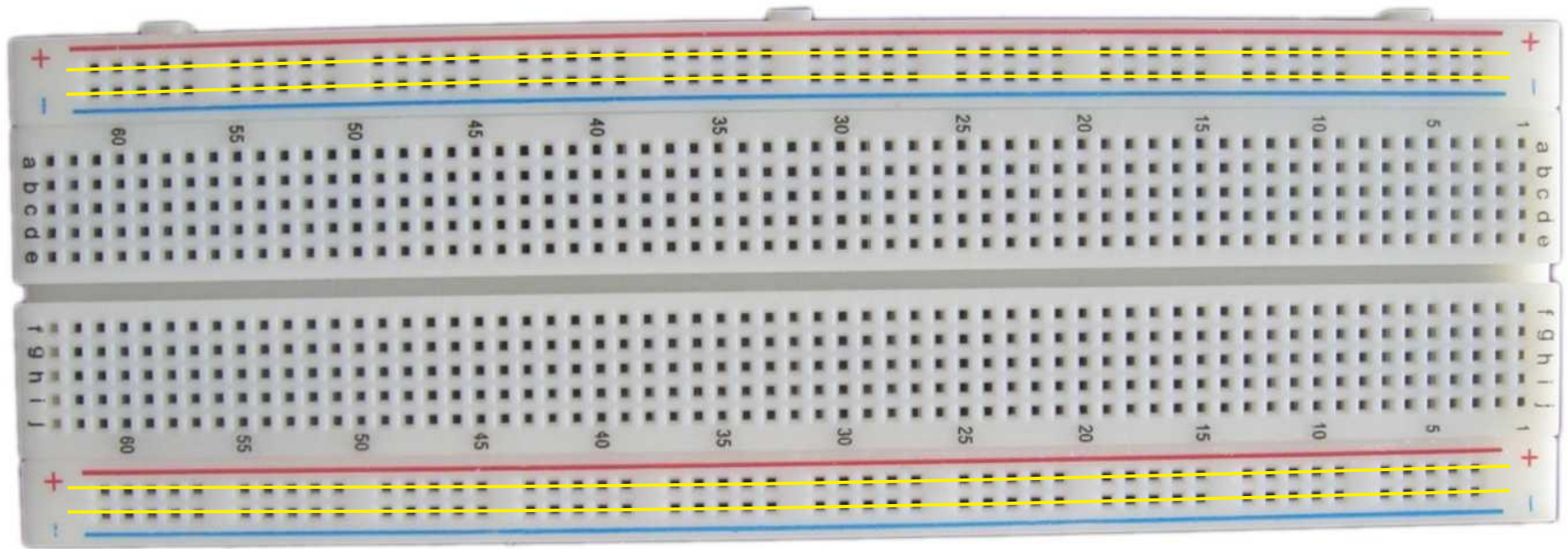
La planche à pain (breadboard) permet de brancher des composants électronique sans soudure (on parle aussi de plaque d'expérimentation)





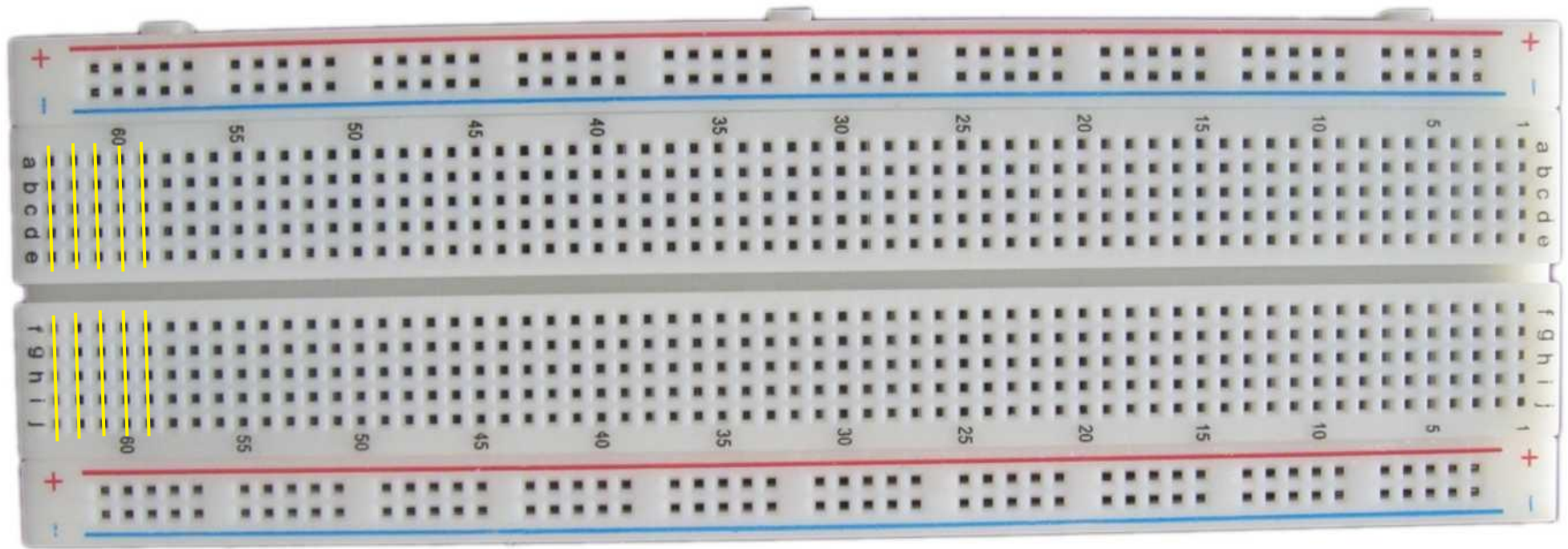
# Interconnexion des trous en interne

Les trous sont liés entre eux en ligne pour les lignes 1, 2, 13 et 14 (marquées + et -)



# Interconnexion des trous en interne

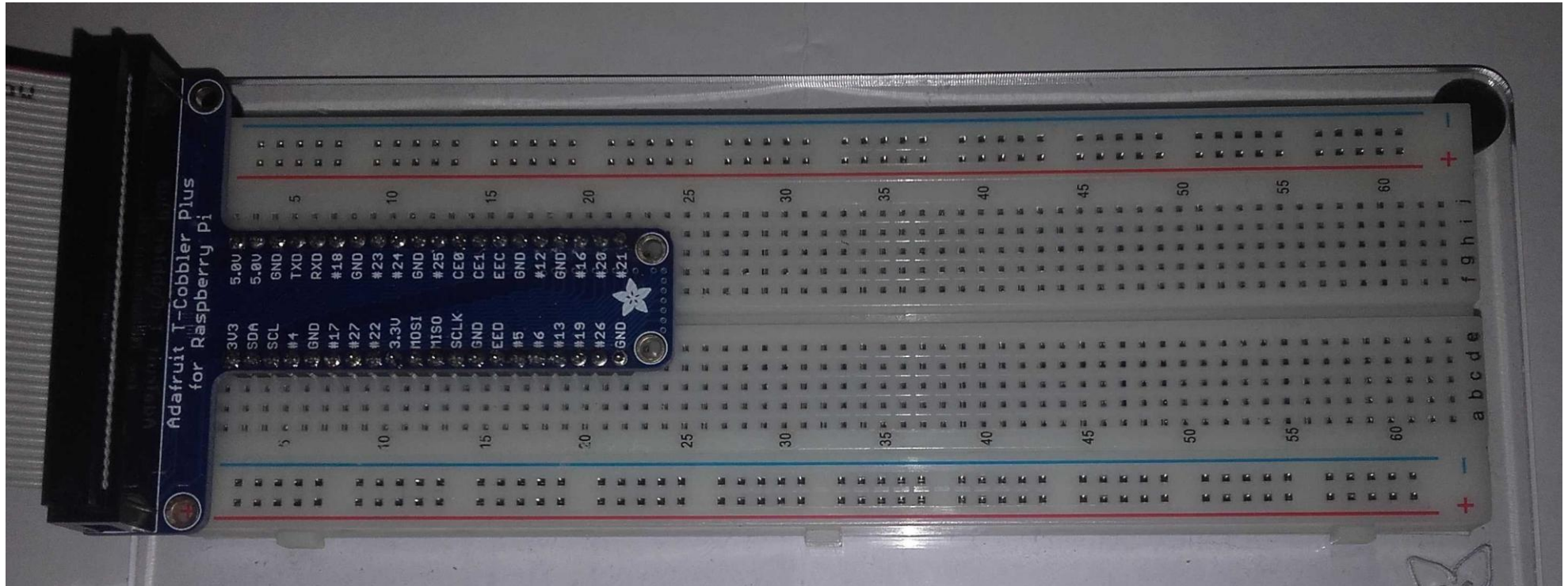
Les autres trous sont reliés en demi-colonne





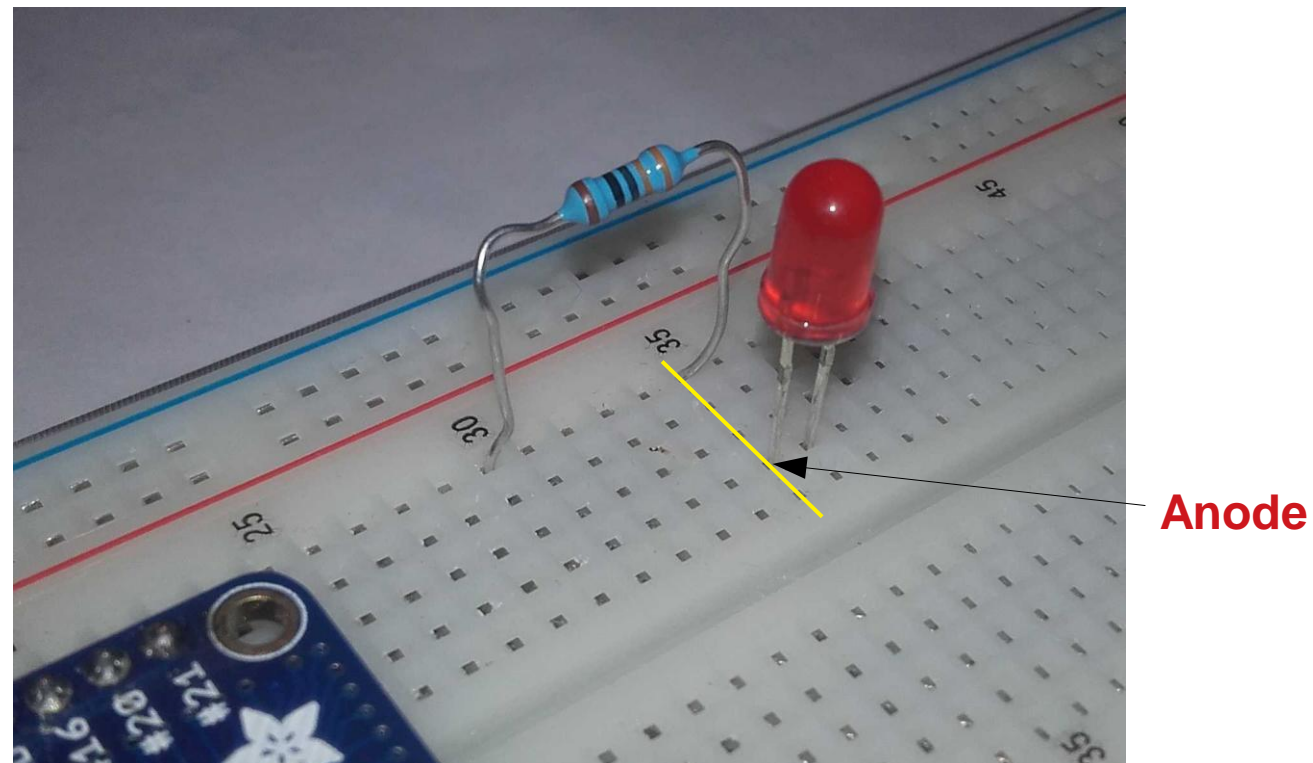
# Le T-Cobbler sous la planche à pain

Le T-Cobbler (avec la numérotation BMC) est branché sur la planche à pain, elle est un report du GPIO du Raspberry Pi afin de faciliter les montages électronique.



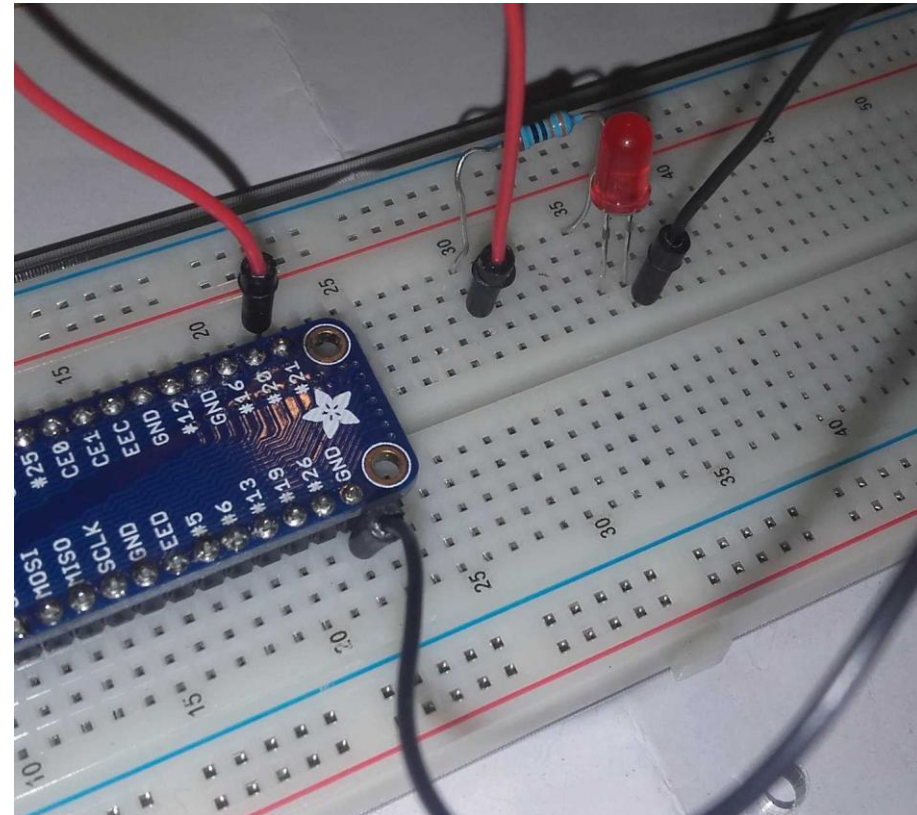
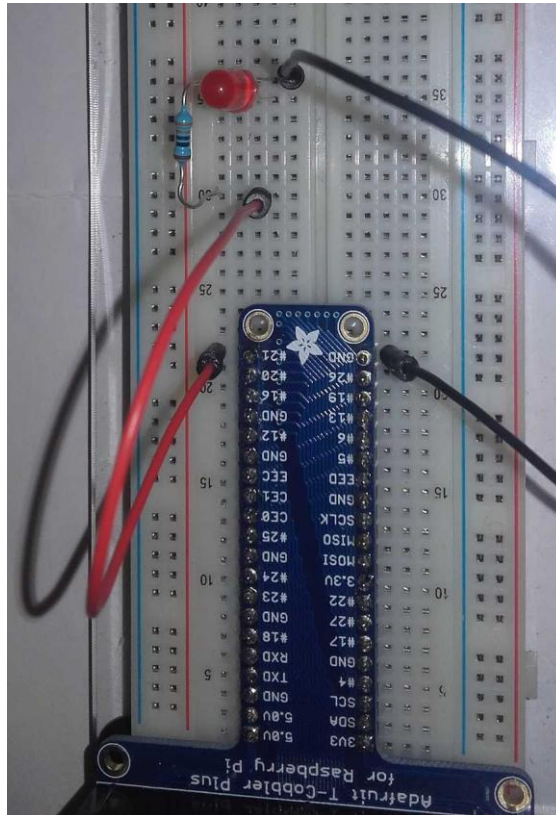
## Branchement de la résistance et de la led

On commence par brancher la résistance et la led rouge en mettant l'anode sur la même colonne qu'une des pattes de la résistance.



## Ajout des câbles pour ferme le circuit

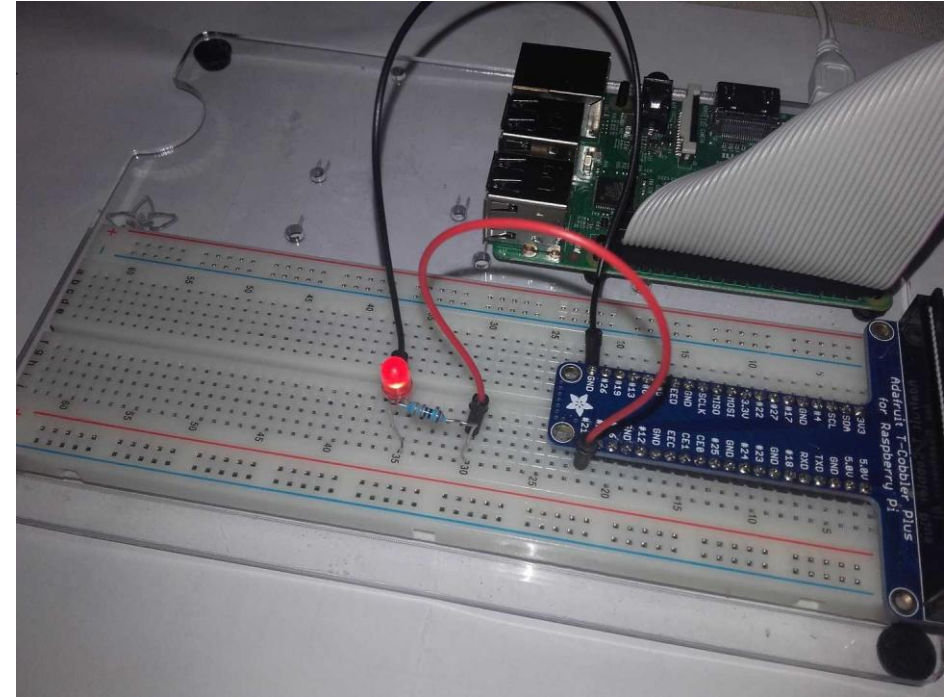
On ajoute ensuite les fils de connexion (jumper cable, wire) pour relier la seconde patte de la résistance à la pin BCM #21 et la cathode à une pin GND.





# Fonctionnement du dispositif

On exécute le script et on constate que la diode s'allume puis qu'elle s'éteint lorsqu'on frappe une touche.

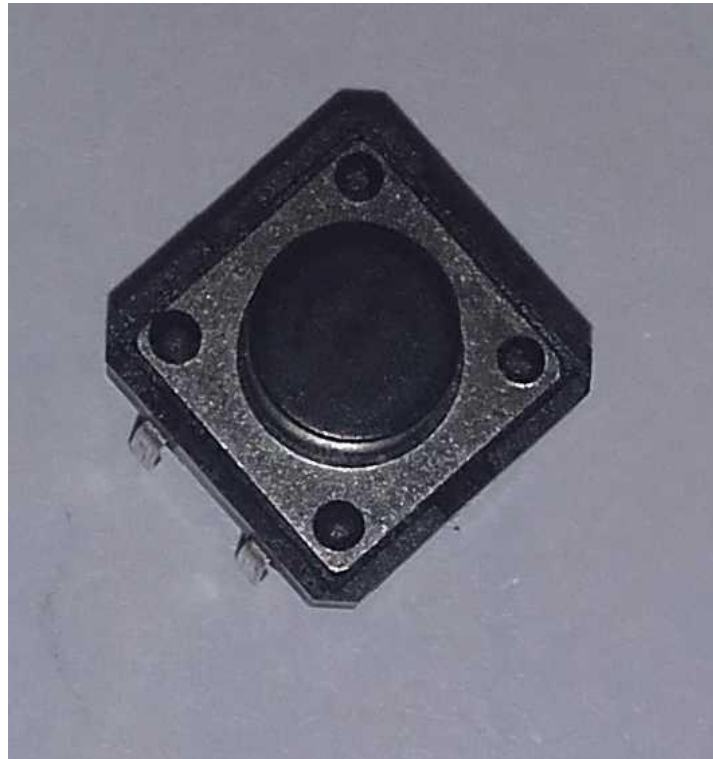


On est donc capable de contrôler l'état d'un pin depuis un programme Python, et par voie de conséquence d'activer des dispositifs électronique.

**La lecture de  
l'état d'entrée  
d'un pin pour  
gérer un  
interrupteur**

## Le bouton poussoir à utiliser

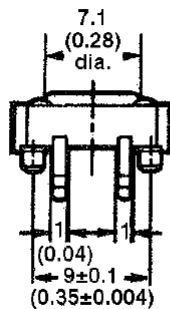
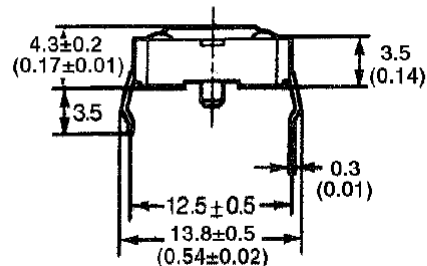
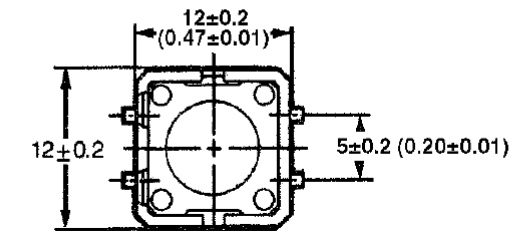
On utilise maintenant un interrupteur à 4 pattes tel que celui ci-dessous afin de contrôler le fonctionnement d'un script Python



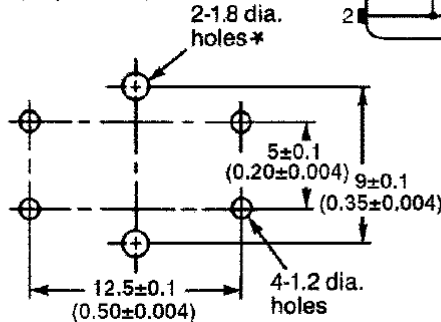
# Extrait du datasheet

Pour comprendre, on doit analyser le document décrivant le produit (le datasheet)

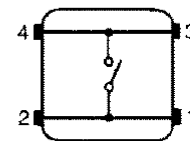
## ■ B3F-4000/-5000/-4005/-5001



PCB mounting  
(top view)

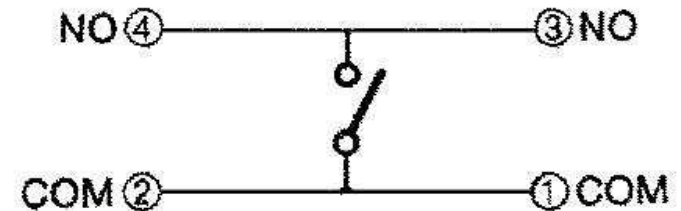


Terminal  
arrangement/  
internal  
connections  
(top view)



## ■ Contact Form

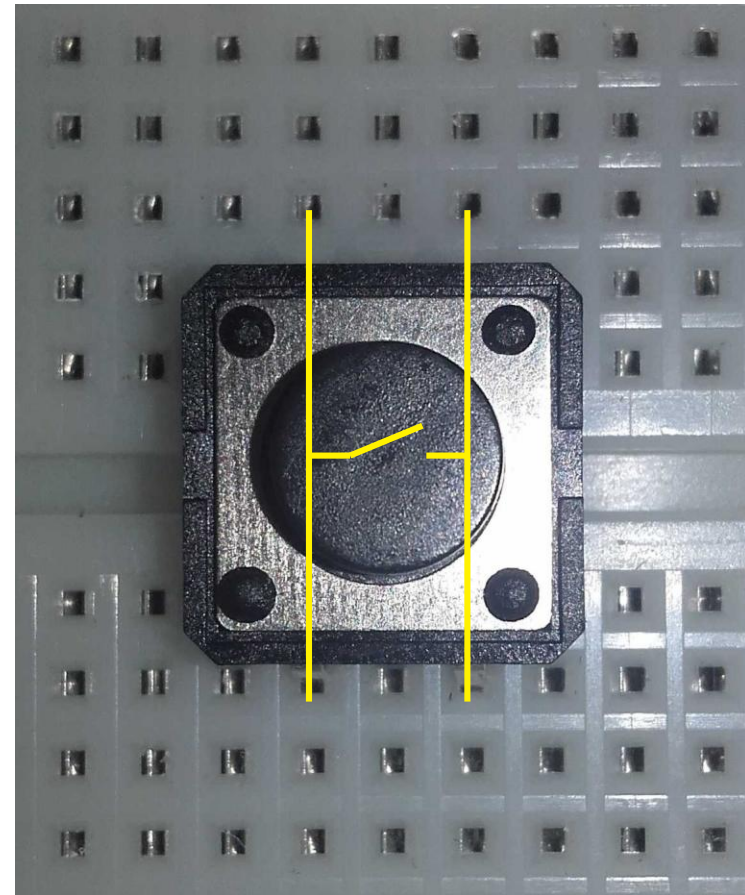
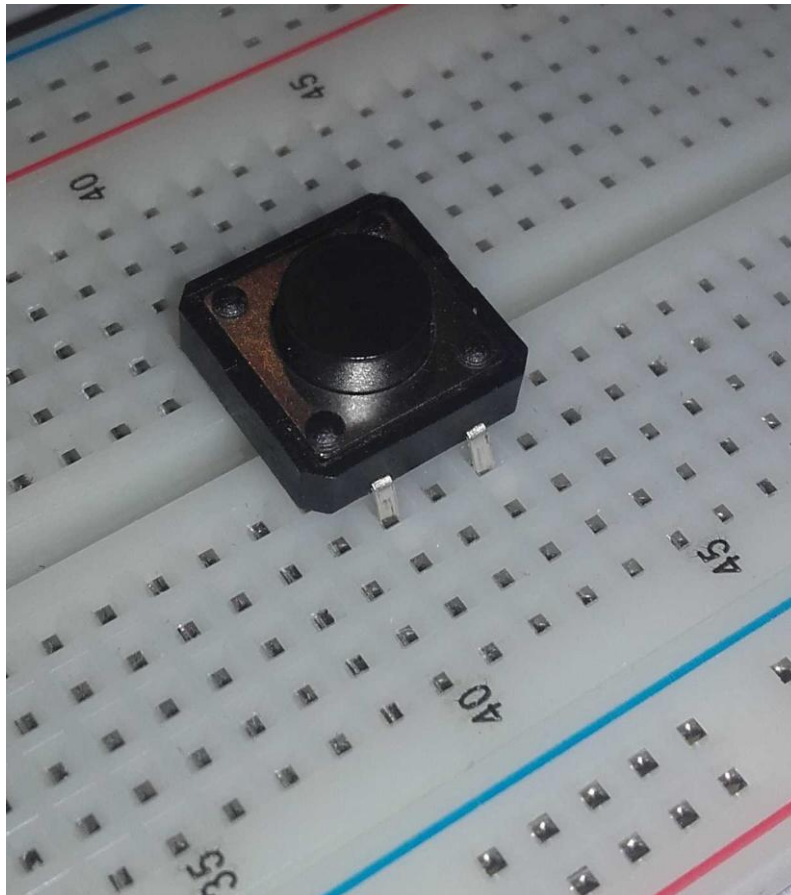
B3F-10□□  
-40□□  
-50□□  
-10□□-G



Voir [https://boutique.semageek.com/fr/index.php?controller=attachment&id\\_attachment=213](https://boutique.semageek.com/fr/index.php?controller=attachment&id_attachment=213)

## Branchement de l'interrupteur

On peut alors brancher cet interrupteur de la manière suivante sur la planche à pain





## Le problème de la patte flottante

Dans une première approche, on peut imaginer brancher l'interrupteur entre une pin 3v3 et la pin BCM #24.

Quand l'interrupteur est fermé, l'entrée BCM #24 est à 3,3 volt : elle est donc considérée comme à l'état haut.

Quand l'interrupteur est ouvert, l'état de l'entrée BCM #24 est alors considérée comme « **flottant** », entre l'état haut et l'état bas, à cause des perturbations électromagnétiques.

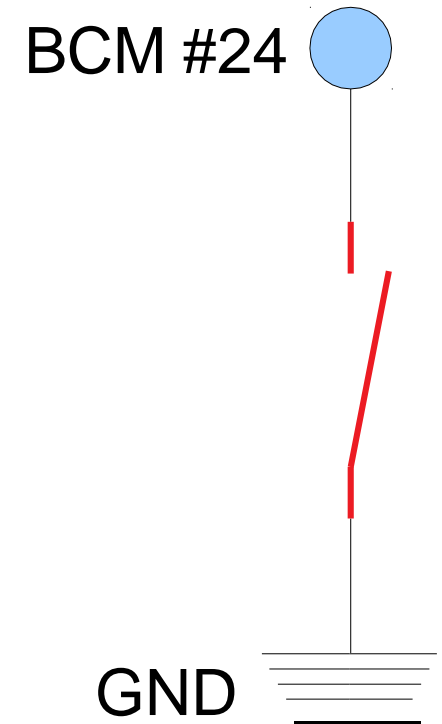


## Le problème de la patte flottante

On peut également faire le même constat si on place l'interrupteur entre une pin GND et la pin BCM #24.

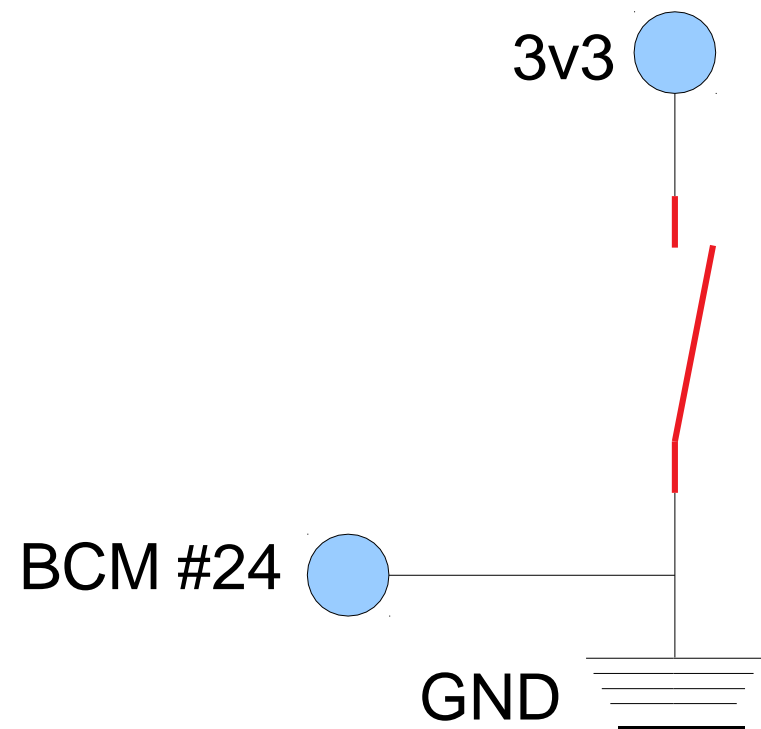
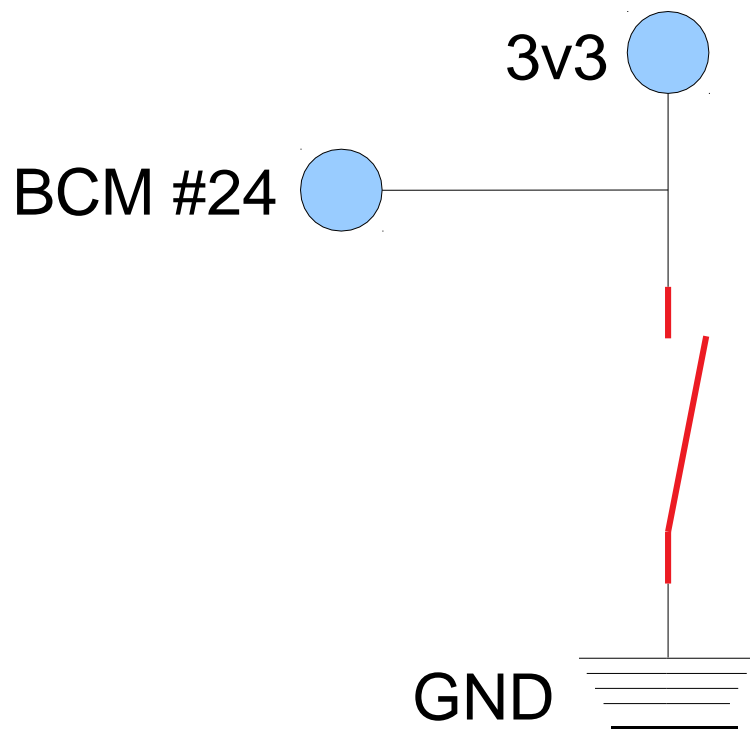
Quand l'interrupteur est fermé, l'entrée BCM #24 est à 0 volt : elle est donc considérée comme à l'état bas.

Quand l'interrupteur est ouvert, l'état de l'entrée BCM #24 est de nouveau considérée comme « **flottant** », donc indéfini entre l'état haut et l'état bas, toujours à cause de perturbations électromagnétiques.



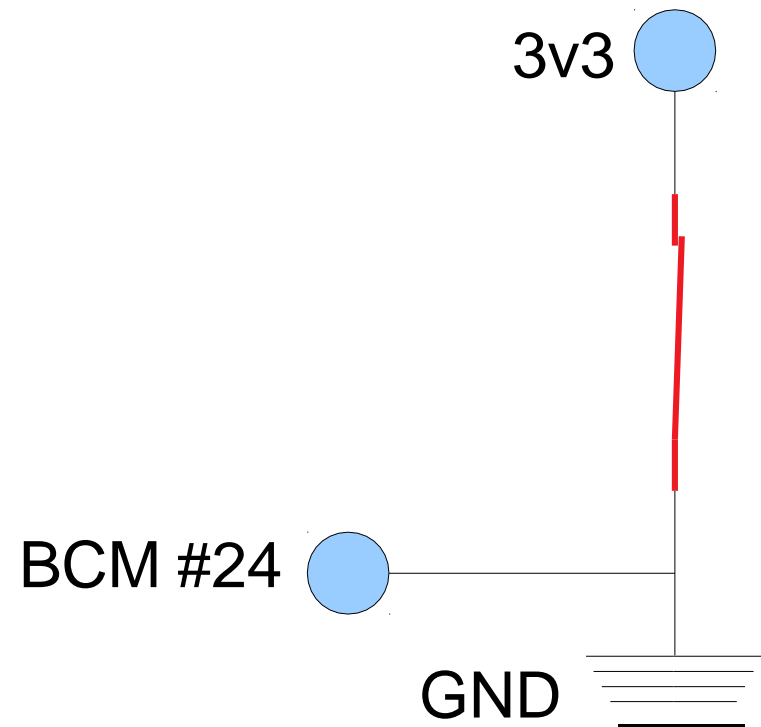
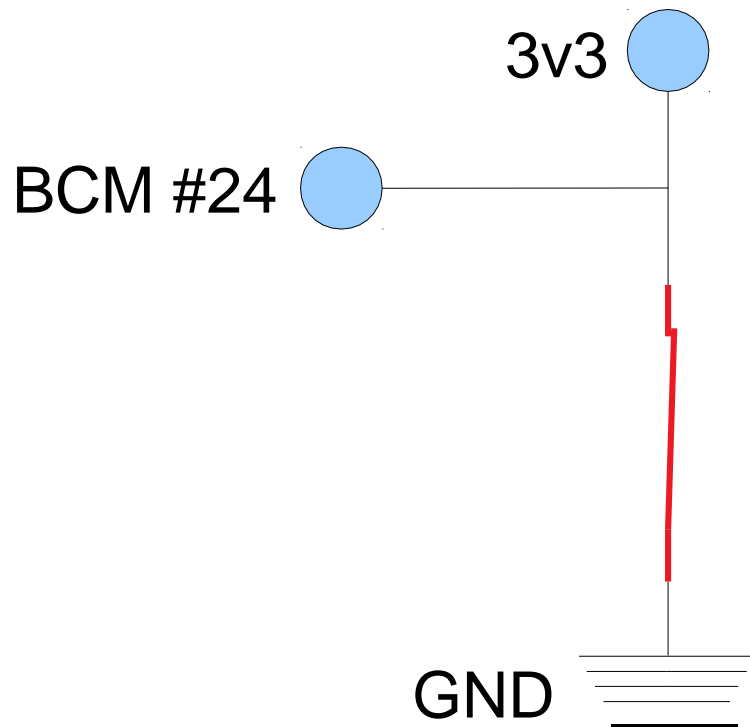
# Tentative naïve de résolution du problème

Pour corriger ce problème de flottement, on va relier le pin BCM #24 au pin 3v3 ou GND de manière à avoir un état stable quand l'interrupteur est ouvert. On a alors l'un des deux montages ci-dessous.

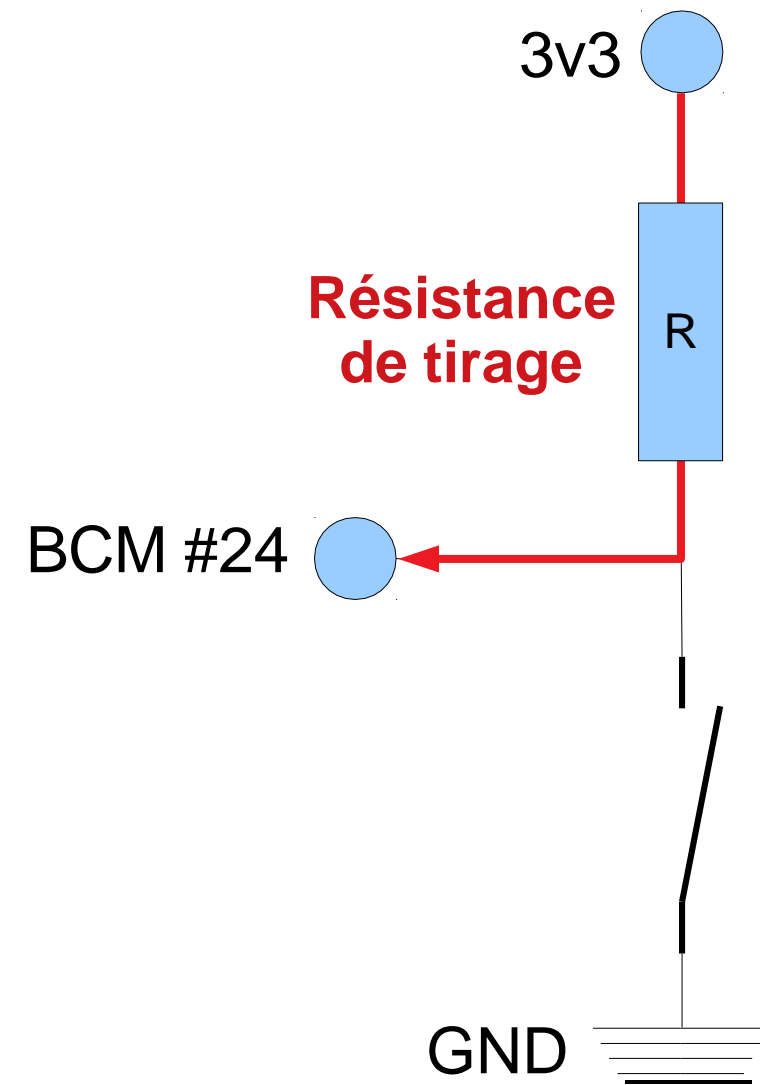


# Tentative naïve de résolution du problème

Malheureusement, dans les deux cas, lorsqu'on ferme l'interrupteur, on obtient un **court-circuit** qui peut être de nature à endommager le Raspberry Pi : on doit ajouter une **résistance**.



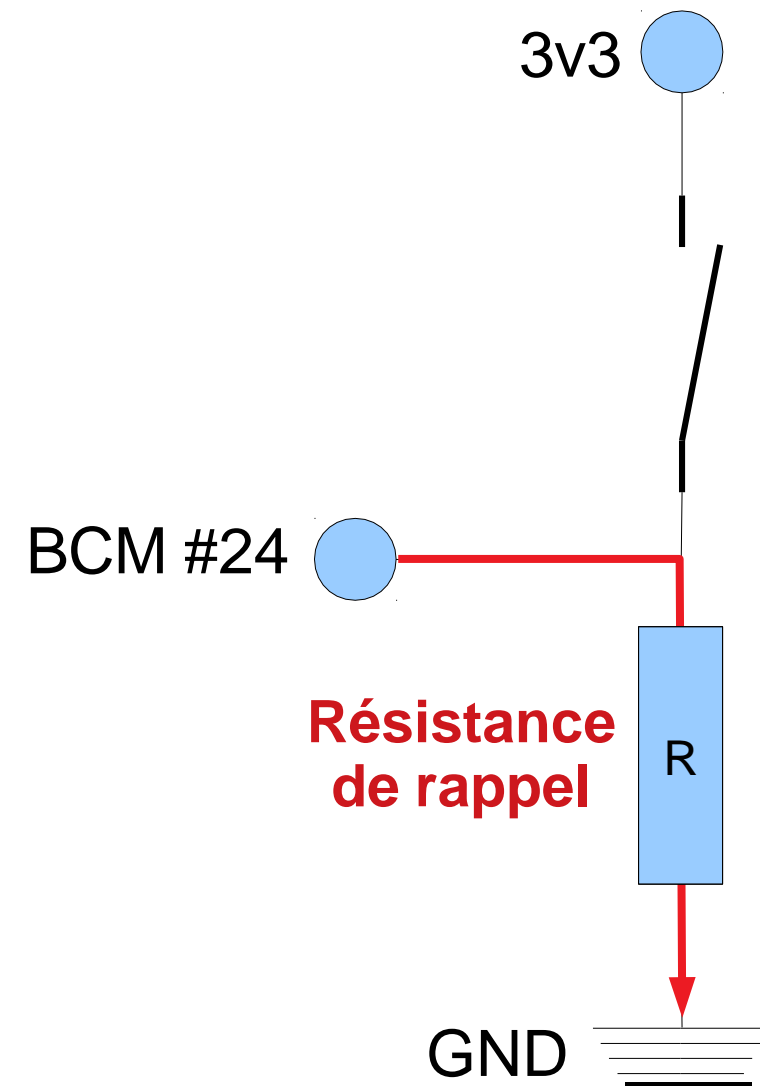
## La résistance de tirage



Dans le premier cas, on ajoute une résistance dite « de tirage » (« pull-up » en anglais) entre les pins 3v3 et BCM #24 afin de fixer BCM #24 à l'état haut lorsque l'interrupteur est ouvert.

Elle permettra aussi de limiter la quantité de courant qui circule entre les 2 pins si BCM #24 est configuré par erreur en sortie et positionné à l'état bas.

## La résistance de rappel

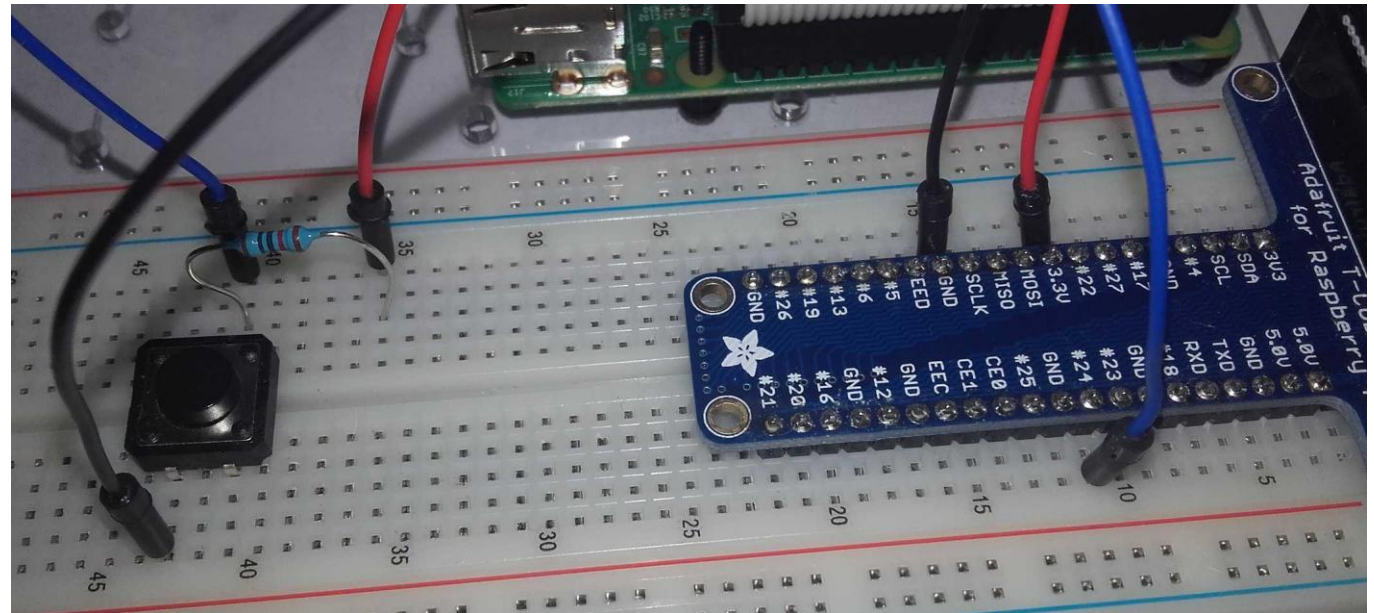
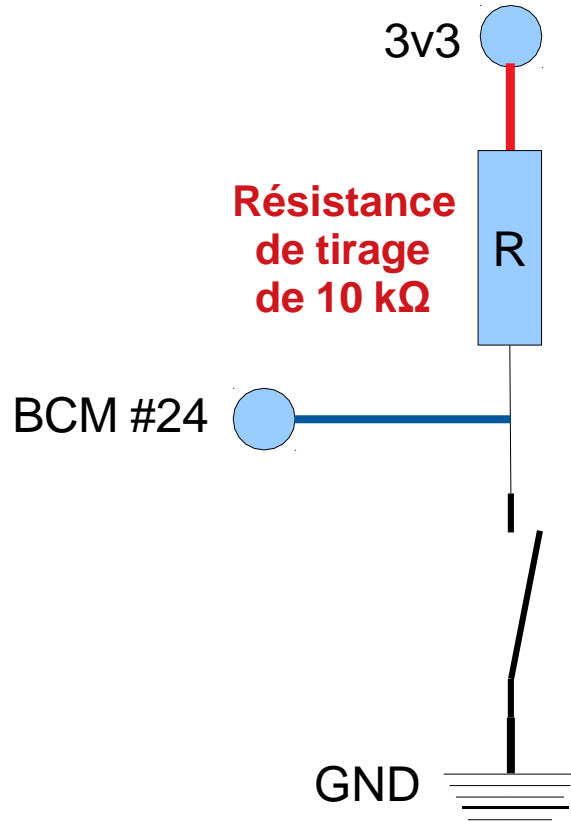


Dans le second cas, on ajoute une résistance dite « de rappel » (« pull-down » en anglais) entre les pins GND et BCM #24 afin de fixer BCM #24 à l'état haut lorsque l'interrupteur est ouvert.

Elle permettra aussi de limiter la quantité de courant qui circule entre les 2 pins si BCM #24 est configuré par erreur en sortie et positionné à l'état haut.

## Première version du circuit

La plupart des tutoriels préconise une valeur de  $10\text{ k}\Omega$ . On utilise donc cette valeur pour effectuer le montage ci-dessous correspondant au schéma de gauche.



## Première version du circuit

On exécute le script ci-dessous :

```
#!/usr/bin/python3

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup (24, GPIO.IN)

while True:

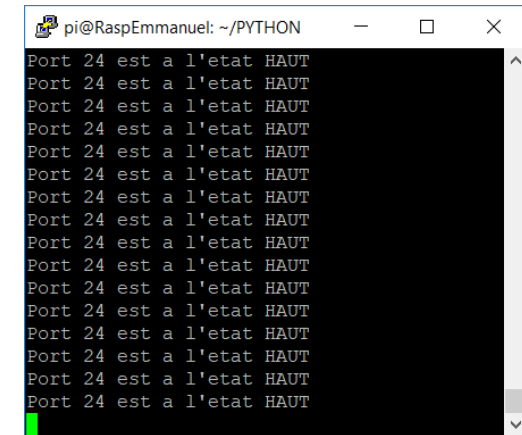
    if GPIO.input(24):

        print ("Port 24 est a l'etat HAUT")

    else:

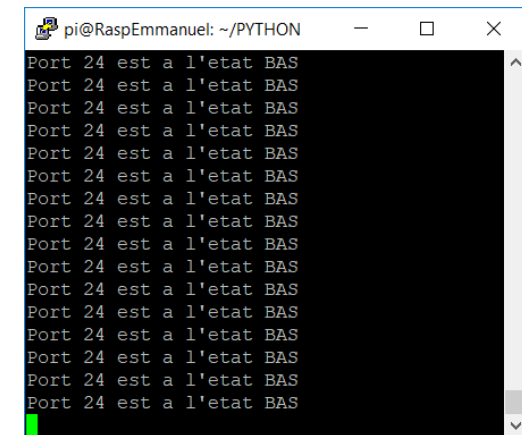
        print ("Port 24 est a l'etat BAS")

GPIO.cleanup()
```

A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' showing the output of the script. The output consists of 15 lines, all of which are 'Port 24 est a l'etat HAUT'. The window has a black background and white text. A green cursor is visible at the bottom left of the terminal area.

```
pi@RaspEmmanuel: ~/PYTHON
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
```

Le bouton est relâché

A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' showing the output of the script. The output consists of 15 lines, all of which are 'Port 24 est a l'etat BAS'. The window has a black background and white text. A green cursor is visible at the bottom left of the terminal area.

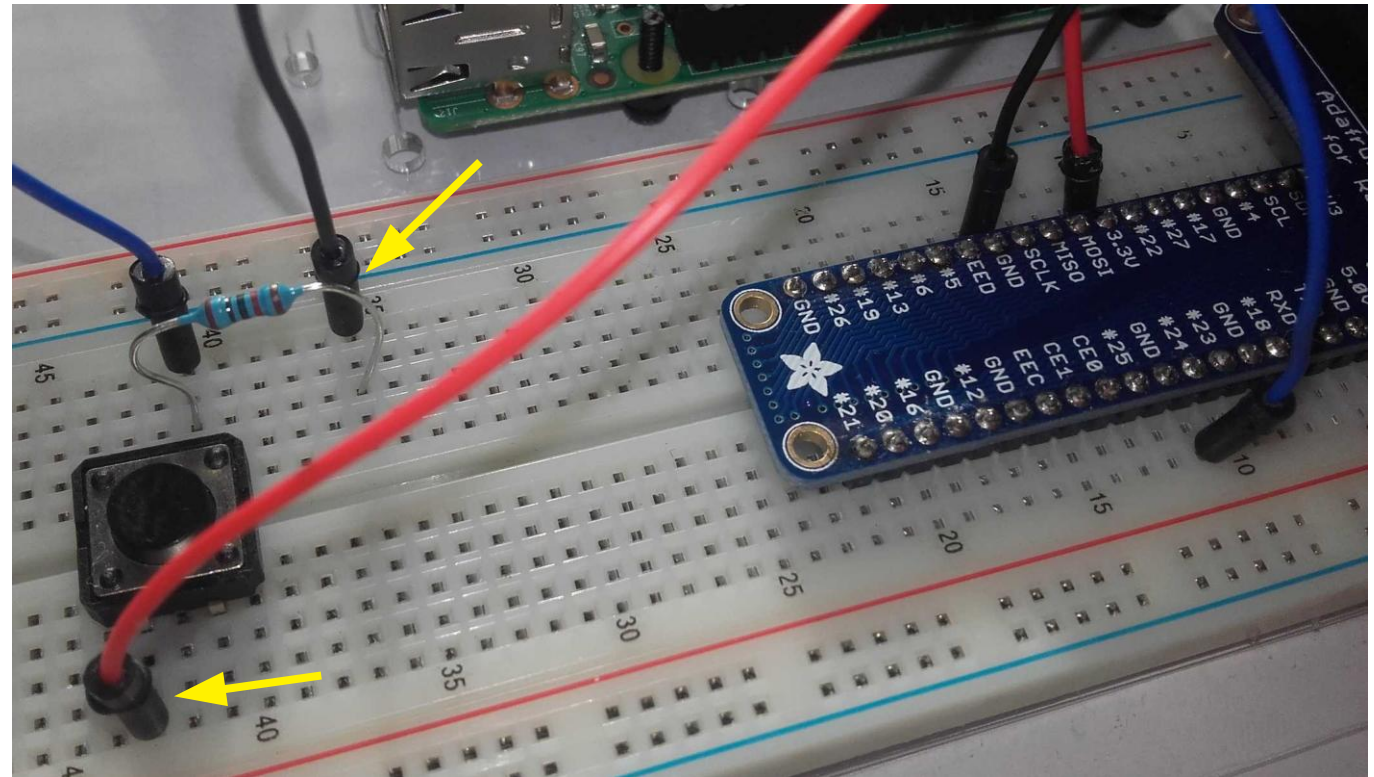
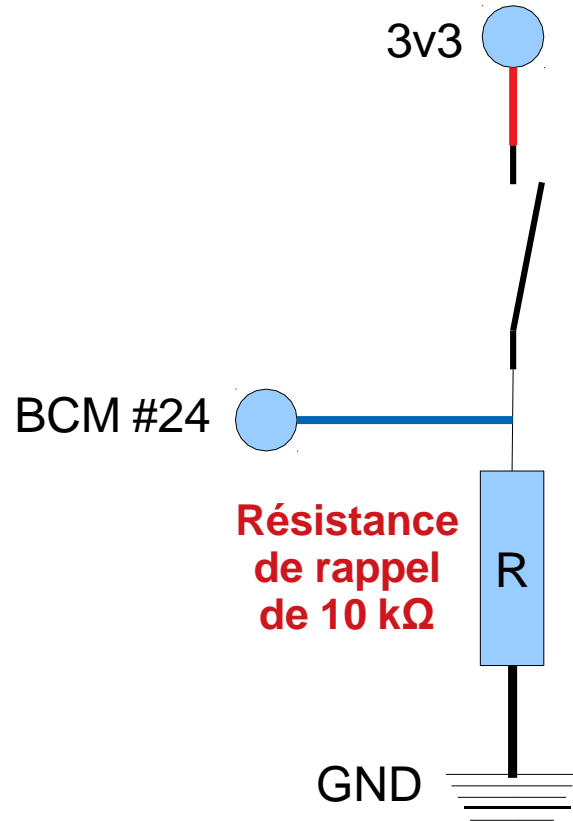
```
pi@RaspEmmanuel: ~/PYTHON
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
```

Le bouton est enfoncé



## Seconde version du circuit

On peut aussi reprendre cette même résistance pour réaliser le second montage. On inverse simplement 2 branchements par rapport au montage précédent.



## Seconde version du circuit

On exécute le même script :

```
#!/usr/bin/python3

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup (24, GPIO.IN)

while True:

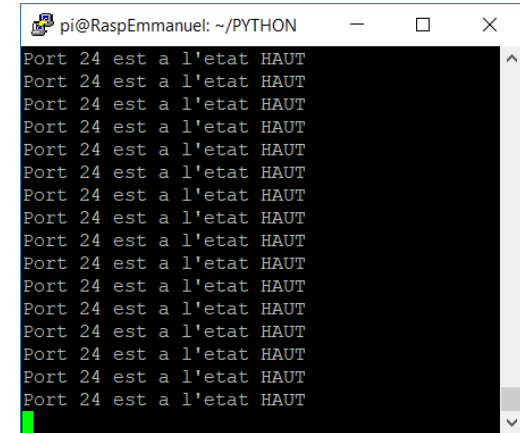
    if GPIO.input(24):

        print ("Port 24 est a l'etat HAUT")

    else:

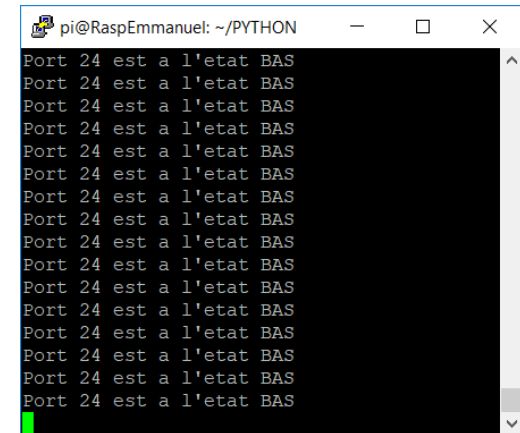
        print ("Port 24 est a l'etat BAS")

GPIO.cleanup()
```

A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' showing the output of the script. The output consists of 15 lines, all reading 'Port 24 est a l'etat HAUT', indicating that the button is being held down. A green cursor is visible at the bottom of the terminal.

```
pi@RaspEmmanuel: ~/PYTHON
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
```

Le bouton est **enfoncé**

A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' showing the output of the script. The output consists of 15 lines, all reading 'Port 24 est a l'etat BAS', indicating that the button has been released. A green cursor is visible at the bottom of the terminal.

```
pi@RaspEmmanuel: ~/PYTHON
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
```

Le bouton est **relâché**

## L'inconvénient du pooling

Le script précédent interroge de façon continue l'état de l'entrée : on parle de **pooling**.

Cette solution présente l'inconvénient de bloquer le programme dans une boucle d'attente active qui provoque le **gaspillage** du temps CPU.

# Ajout d'une temporisation dans le script

On peut diminuer la cadence d'interrogation en introduisant une temporisation grâce à l'instruction `time.sleep`.

```
#!/usr/bin/python3

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup (24, GPIO.IN)

while True:

    if GPIO.input(24):
        print ("Port 24 est a l'etat HAUT")
    else:
        print ("Port 24 est a l'etat BAS")

    time.sleep (.010)

GPIO.cleanup()
```

## Attente d'un front

Une autre solution consiste à **bloquer cette boucle** en attendant la survenue d'un événement grâce à l'instruction `GPIO.wait_for_edge` qui prend les paramètres suivants :

- ❑ le numéro du pin concerné ;
- ❑ le type de front attendu :
  - ❑ `GPIO.RISING` (pour un front montant) ;
  - ❑ `GPIO.FALLING` (pour un front descendant) ;
  - ❑ `GPIO.BOTH` (pour les deux types de front) ;
- ❑ éventuellement un *timeout*, exprimé en millisecondes au-delà duquel, la fonction arrête d'attendre l'événement.

Voir <https://sourceforge.net/p/raspberry-gpio-python/wiki/Inputs/>

# Attente d'un front

Avec cette nouvelle instruction, on peut alors développer le script ci-dessous.

```
#!/usr/bin/python3

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setup (24, GPIO.IN)

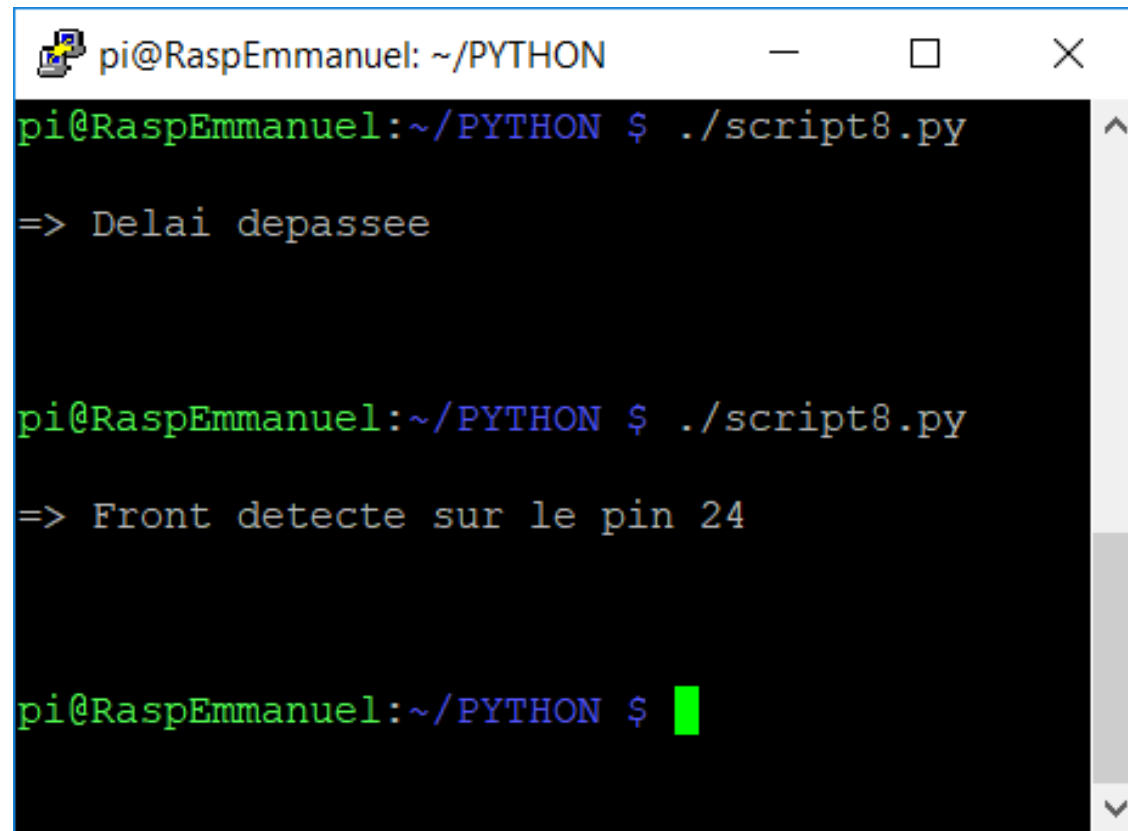
channel = GPIO.wait_for_edge (24, GPIO.RISING, timeout=5000)

if channel is None:
    print('\n=> Delai depassee\n\n\n')
else:
    print('\n=> Front detecte sur le pin', channel, '\n\n\n')

GPIO.cleanup()
```

## Attente d'un front

On exécute le script une première fois, sans enfoncer le bouton, puis une seconde fois en enfonçant l'interrupteur avant les 5 secondes.

A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' with standard window controls. It shows the execution of a script named 'script8.py' twice. The first execution results in the output '=> Delai depassee'. The second execution results in the output '=> Front detecte sur le pin 24'. The prompt is currently at the third line, ready for input.

```
pi@RaspEmmanuel: ~/PYTHON
pi@RaspEmmanuel:~/PYTHON $ ./script8.py
=> Delai depassee

pi@RaspEmmanuel:~/PYTHON $ ./script8.py
=> Front detecte sur le pin 24

pi@RaspEmmanuel:~/PYTHON $
```

# Programmation événementielle

Le module GPIO nous permet aussi d'associer une fonction de callback à un événement survenant sur un pin grâce à l'instruction `GPIO.add_event_detect` qui prend 3 paramètres :

- ❑ le numéro du pin concerné ;
- ❑ le type d'événement à surveiller :
  - ❑ `GPIO.RISING` (un front montant) ;
  - ❑ `GPIO.FALLING` (un front descendant) ;
  - ❑ `GPIO.BOTH` (un front montant ou descendant) ;
- ❑ la fonction de callback à appeler, qui doit prendre un paramètre qui se verra assigner le numéro du pin sur lequel survient l'événement).



# Programmation événementielle

Cette nouvelle possibilité est plus intéressante que les deux solutions précédente car elle permet de créer des programmes **réagissant à des événements**. On écrit donc un nouveau script qu'on fait fonctionner avec le dernier montage.

```
#!/usr/bin/python3

# coding=utf-8

import RPi.GPIO as GPIO
import datetime

def my_callback (channel):
    if GPIO.input(24) == GPIO.HIGH:
        print('\nFront montant a ' + str(datetime.datetime.now()))
    else:
        print('\nFront descendant a ' + str(datetime.datetime.now()))
```

# Programmation événementielle

```
try:
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(24, GPIO.IN)

    GPIO.add_event_detect(24, GPIO.BOTH, callback=my_callback)

    message = input('\nAppuyez sur une touche pour quitter.\n')

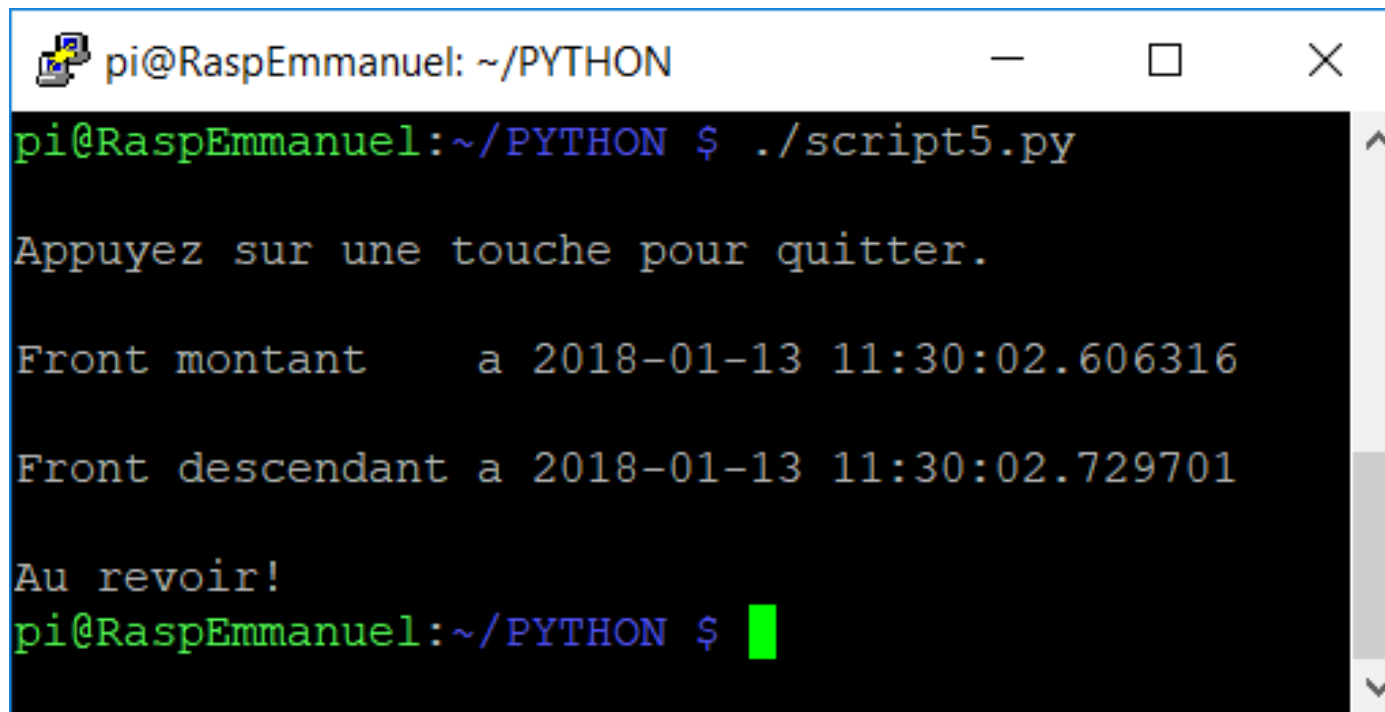
finally:
    GPIO.cleanup()

print("Au revoir!")
```

Voir <https://grantwinney.com/using-pullup-and-pulldown-resistors-on-the-raspberry-pi/>  
<http://raspi.tv/2013/how-to-use-interrupts-with-python-on-the-raspberry-pi-and-rpi-gpio-part-3>  
<http://raspi.tv/2014/rpi-gpio-update-and-detecting-both-rising-and-falling-edges>  
<https://pypi.python.org/pypi/RPi.GPIO>  
<https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/?SetFreedomCookie>

# Programmation événementielle

Lorsqu'on exécute le script, on obtient un résultat tel que celui ci-dessous lorsqu'on enfonce puis relâche le bouton.

A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' with standard window controls. The terminal shows the execution of './script5.py'. The output includes a prompt to press a key to quit, followed by two lines of sensor data: 'Front montant a 2018-01-13 11:30:02.606316' and 'Front descendant a 2018-01-13 11:30:02.729701'. It ends with 'Au revoir!' and a new command prompt. A green cursor is visible at the end of the last prompt line.

```
pi@RaspEmmanuel: ~/PYTHON
pi@RaspEmmanuel:~/PYTHON $ ./script5.py

Appuyez sur une touche pour quitter.

Front montant      a 2018-01-13 11:30:02.606316

Front descendant a 2018-01-13 11:30:02.729701

Au revoir!
pi@RaspEmmanuel:~/PYTHON $
```

## Supprimer une gestion d'événement

Pour supprimer l'association entre un événement sur un pin et une fonction de callback, on utilise l'instruction `GPIO.remove_event_detect` en passant en paramètre le numéro du pin concerné.

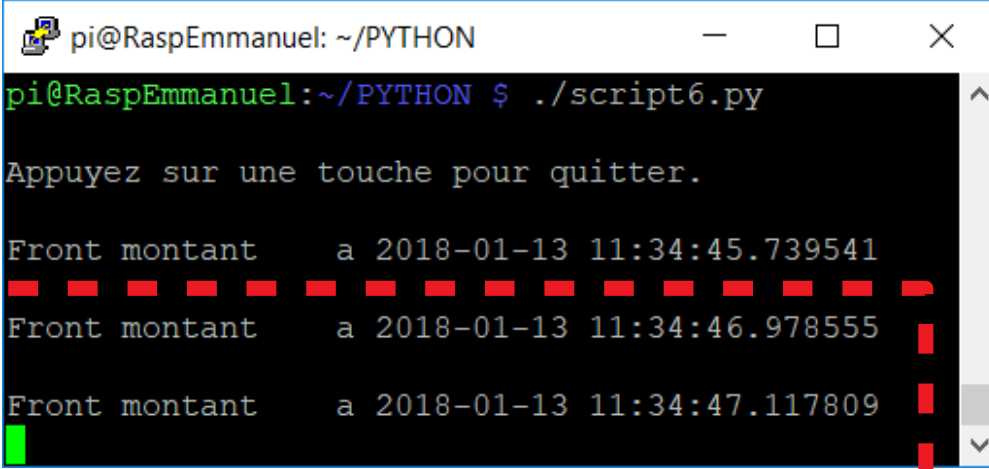
Exemple :

```
GPIO.remove_event_detect (24)
```

# Le problème du rebond de l'interrupteur

On modifie le script que le front montant. On le lance et on peut parfois constater quelques « hoquets » dans le traitement...

En effet, dans l'exemple ci-contre, le second appui sur le bouton a provoqué **2** appels à la fonction de callback

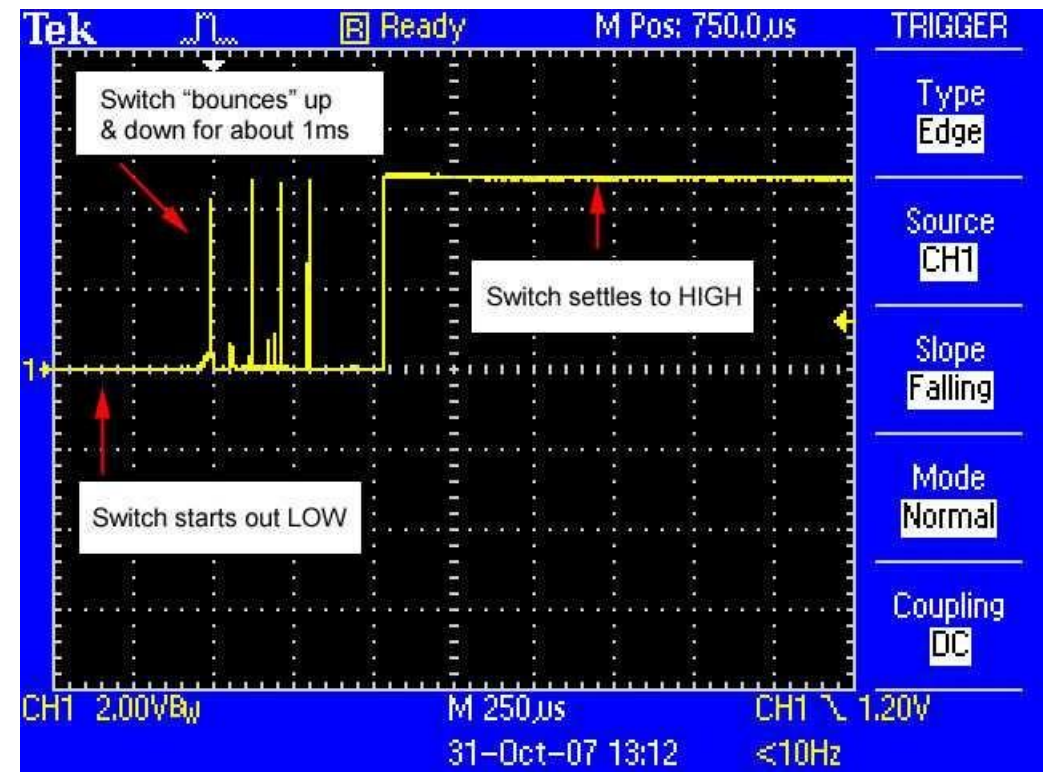


```
pi@RaspEmmanuel: ~/PYTHON
pi@RaspEmmanuel:~/PYTHON $ ./script6.py
Appuyez sur une touche pour quitter.
Front montant      a 2018-01-13 11:34:45.739541
Front montant      a 2018-01-13 11:34:46.978555
Front montant      a 2018-01-13 11:34:47.117809
```

# Le problème du rebond de l'interrupteur

Ce désagrément est dû au fonctionnement de l'interrupteur, qui est, comme tout dispositif, **imparfait**.

En effet, cet interrupteur peut provoquer des micro-rebonds pendant une courte phase de transition qui peuvent être perçus par le Raspberry Pi comme autant de changements d'état fantômes.



Voir <https://arduino103.blogspot.fr/2011/05/entree-bouton-resistance-pull-up-pull.html>  
<http://www.ladyada.net/learn/arduino/lesson5.html>

## 2 types de solutions

Pour contourner ce problème, on peut utiliser deux solutions (voire le combiner) :

- Une solution **logicielle** qui consiste à introduire une **temporisation** dans le programme
- Une solution **matérielle** qui s'appuie sur l'utilisation d'un **condensateur**, placé en parallèle de l'interrupteur à « fiabiliser ».



# Une première solution logicielle

La solution logicielle va donc simplement consister à enregistrer l'état au début du traitement de l'événement puis à vérifier au bout d'un temps d'environ 10 ms si cet état est toujours le même afin de valider l'événement.

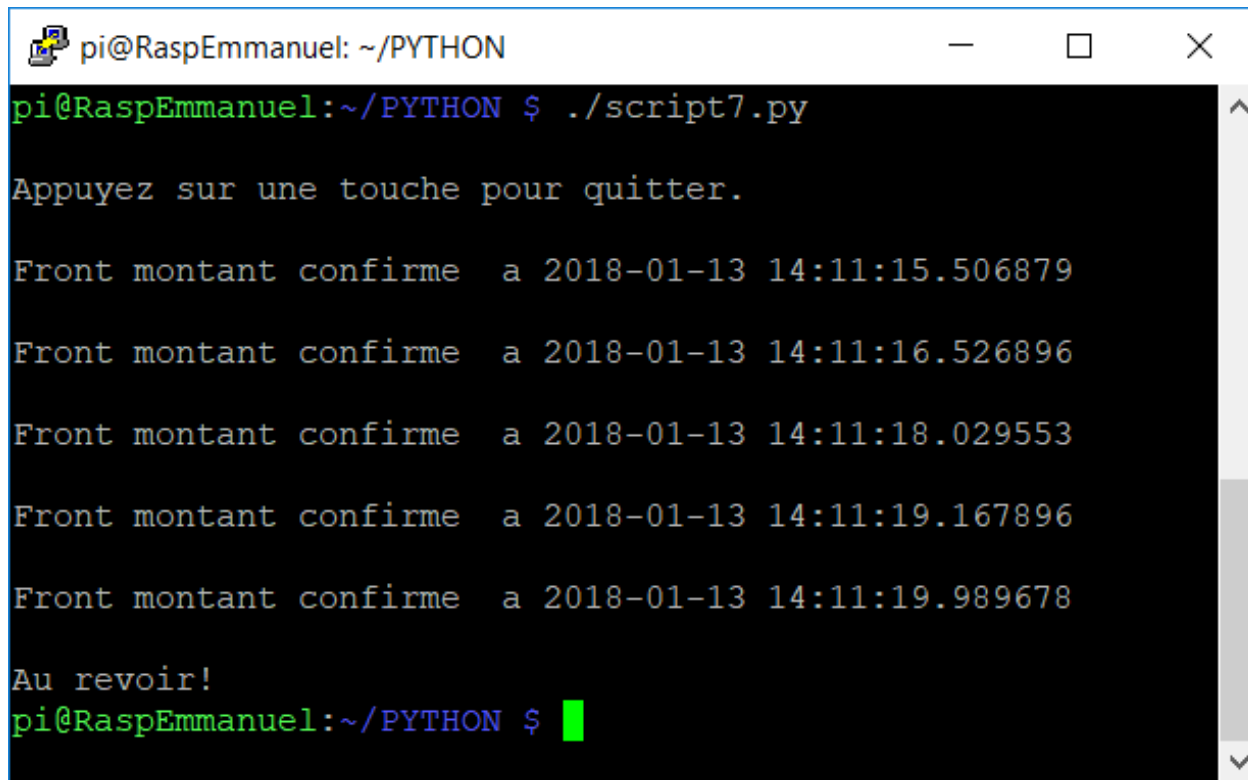
```
import RPi.GPIO as GPIO
import datetime
import time

def my_callback(channel):
    f1 = GPIO.input(24)
    time.sleep (.010)
    f2 = GPIO.input(24)

    if f1 == GPIO.HIGH and f2 == GPIO.HIGH :
        print('\nFront montant confirme a ' + str(datetime.datetime.now()))
```

# Une première solution logicielle

Lorsqu'on exécute ce nouveau script, on constate que le problème semble résolu.

A screenshot of a terminal window titled "pi@RaspEmmanuel: ~/PYTHON". The window shows the execution of a script named "script7.py". The output of the script is as follows:

```
pi@RaspEmmanuel:~/PYTHON $ ./script7.py
Appuyez sur une touche pour quitter.
Front montant confirme a 2018-01-13 14:11:15.506879
Front montant confirme a 2018-01-13 14:11:16.526896
Front montant confirme a 2018-01-13 14:11:18.029553
Front montant confirme a 2018-01-13 14:11:19.167896
Front montant confirme a 2018-01-13 14:11:19.989678
Au revoir!
pi@RaspEmmanuel:~/PYTHON $
```

The terminal window has a standard Linux window title bar with minimize, maximize, and close buttons. The prompt is green, and the command and its output are white on a black background.

## Une seconde solution logicielle

On peut obtenir un résultat similaire en utilisant le paramètre `bouncetime` de l'instruction `GPIO.add_event_detect`.

```
#!/usr/bin/python3

import RPi.GPIO as GPIO
import datetime

def my_callback(channel):
    print('\nFront montant a ' + str(datetime.datetime.now()))

try:
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(24, GPIO.IN)

    GPIO.add_event_detect (24, GPIO.RISING, callback=my_callback, bouncetime=50)

    message = input('\nAppuyez sur une touche pour quitter.\n')

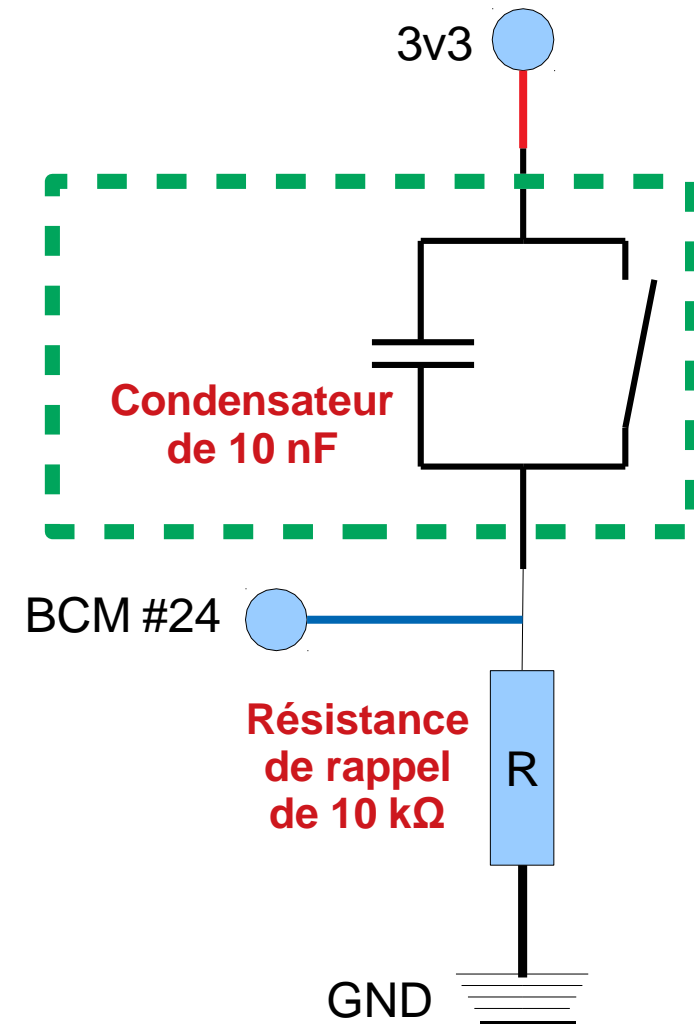
finally:
    GPIO.cleanup()

print("Au revoir!")
```

## La solution matérielle

La solution matérielle consiste à utiliser un condensateur de 10 nF qui va « s'opposer » aux changements rapides engendrés par les rebonds à l'image d'un amortisseur.

Il en résulte un signal moins « perturbé » aux bornes de cet interrupteur « fiabilisé » en pointillés verts.



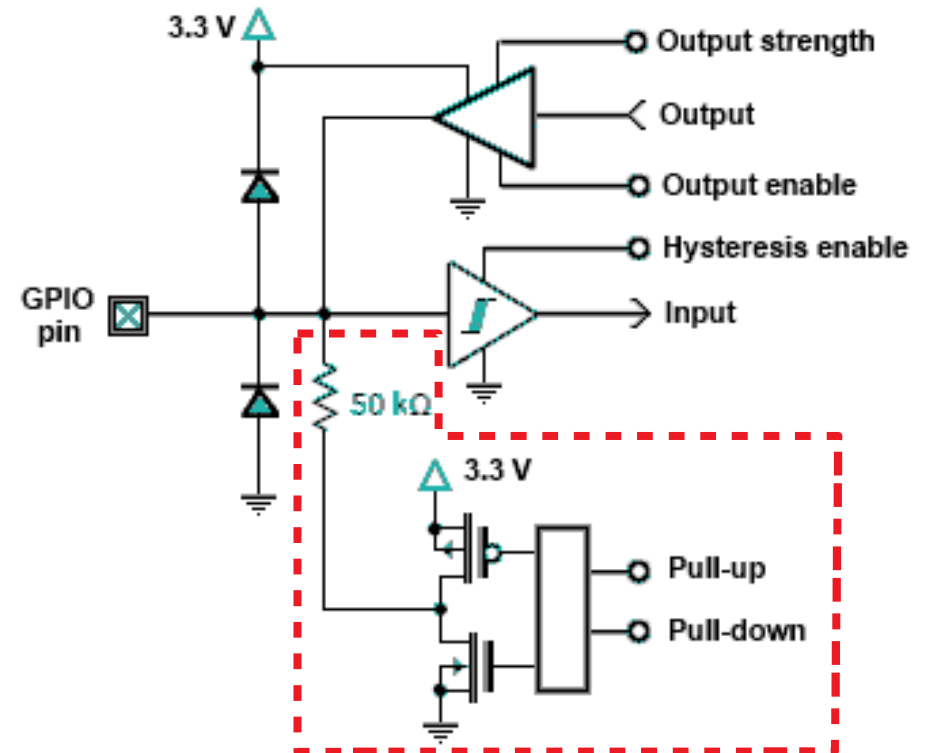
Voir <http://eskimon.fr/96-arduino-204-un-simple-bouton>

# La résistance interne des pins du GPIO

Le fonctionnement des pins du GPIO est complexe et intègre en son sein divers petits dispositifs qui peuvent être activés de façon logicielle.

C'est notamment le cas d'une résistance interne de 50 k pouvant être utilisée pour le pull-down ou le pull-up.

Equivalent Circuit for Raspberry Pi GPIO pins



Voir <http://www.raspberrymd.com/presentations/raspberrymd-windgassen.pdf>  
<http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications>

## Configuration de la résistance interne

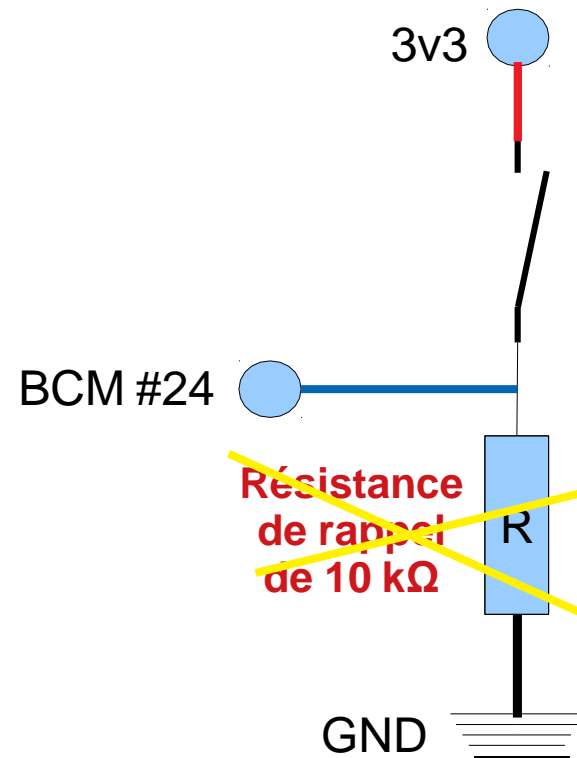
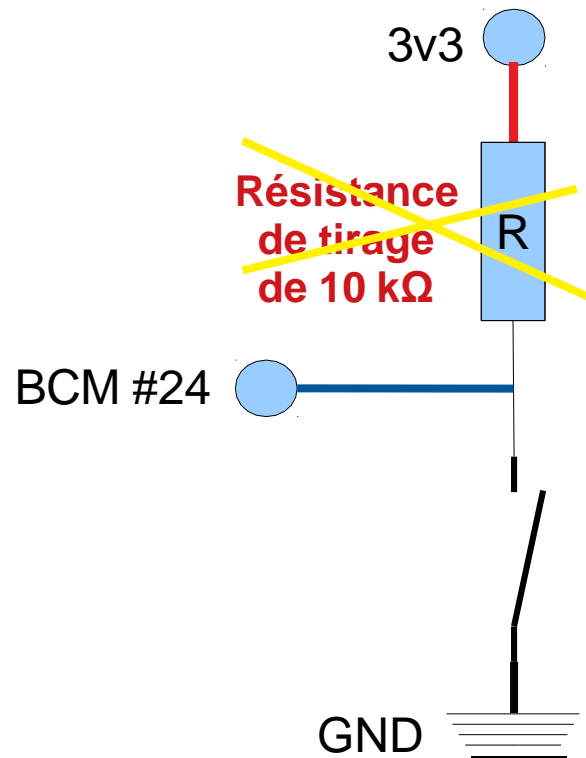
D'un point de vue logiciel, cette caractéristique est spécifiée grâce au paramètre `pull_up_down` de l'instruction `GPIO.setup` qui peut prendre les deux valeurs ci-dessous :

- ❑ `GPIO.PUD_UP` pour utiliser la résistance interne comme un pull-up ;
- ❑ `GPIO.PUD_DOWN` pour utiliser cette même résistance interne comme pull-down.

Voir <https://sourceforge.net/p/raspberry-gpio-python/wiki/Inputs/>  
<https://makezine.com/projects/tutorial-raspberry-pi-gpio-pins-and-python/>

# La simplification des montages

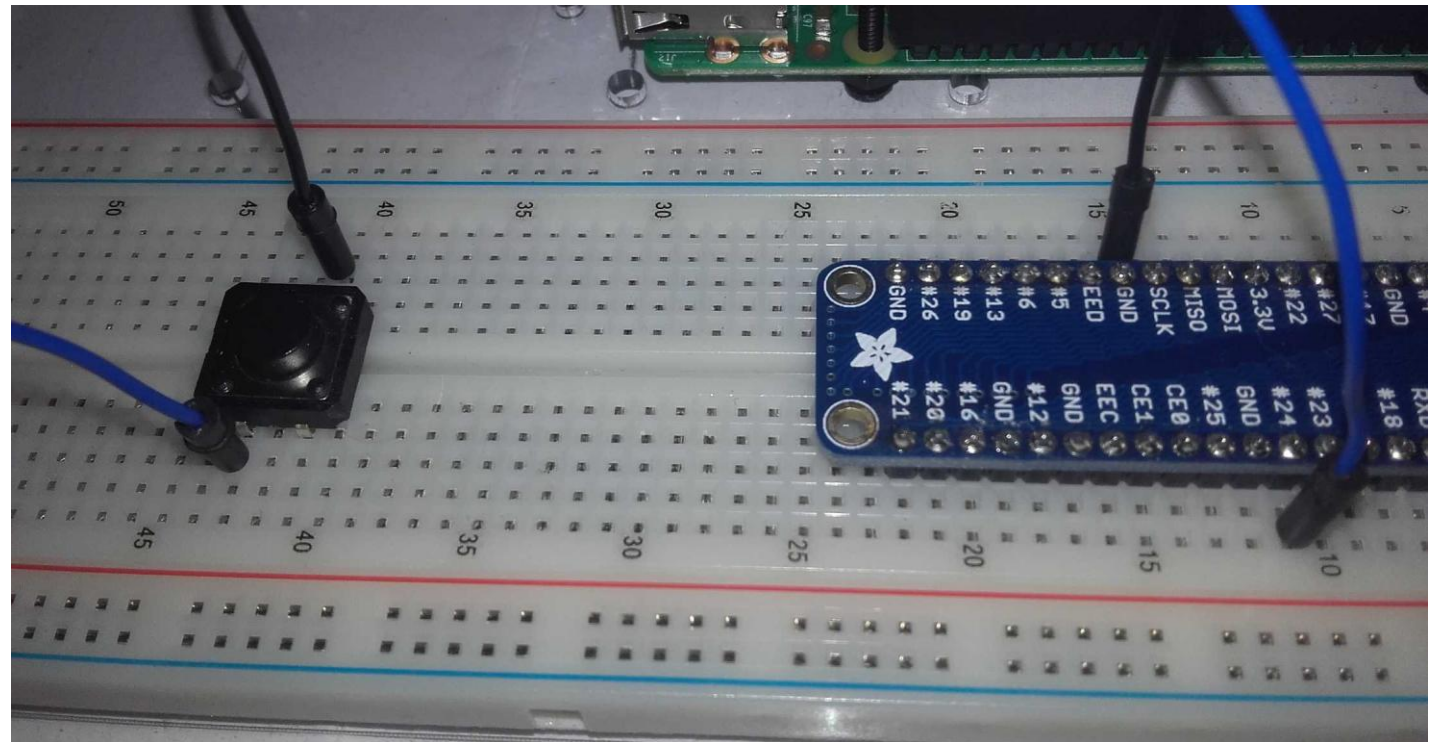
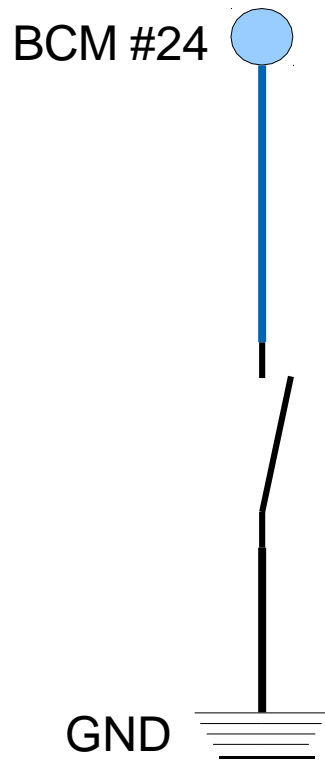
Avec cette nouvelle instruction, on peut simplifier le montage électronique en supprimant la résistance de tirage et de rappel des deux montages que nous avons étudiés.





## Le cas du premier montage

Dans le premier cas, BCM #24 remplace 3v3 et on utilise la résistance interne en mode pull-up



## Le cas du premier montage

On exécute alors le script ci-dessous :

```
#!/usr/bin/python3

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

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

while True:

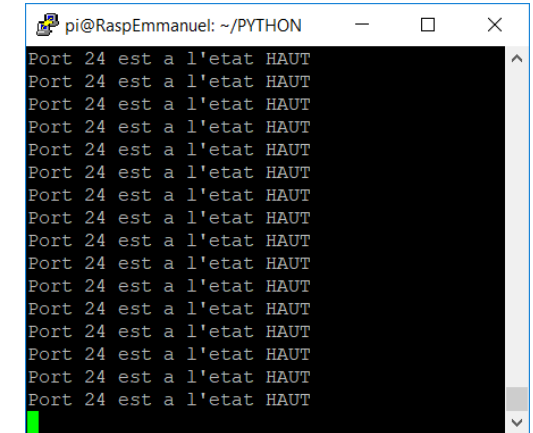
    if GPIO.input(24):

        print ("Port 24 est a l'etat HAUT")

    else:

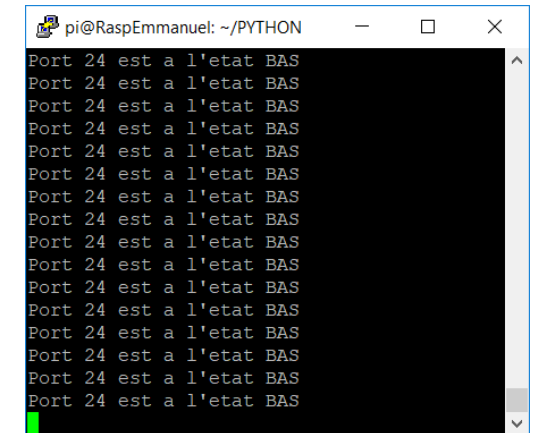
        print ("Port 24 est a l'etat BAS")

GPIO.cleanup()
```

A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' showing the output of the script. It displays 15 lines of text: 'Port 24 est a l'etat HAUT'. The window has a standard Linux terminal interface with a title bar and scrollbars.

```
pi@RaspEmmanuel: ~/PYTHON
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
```

Le bouton est relâché

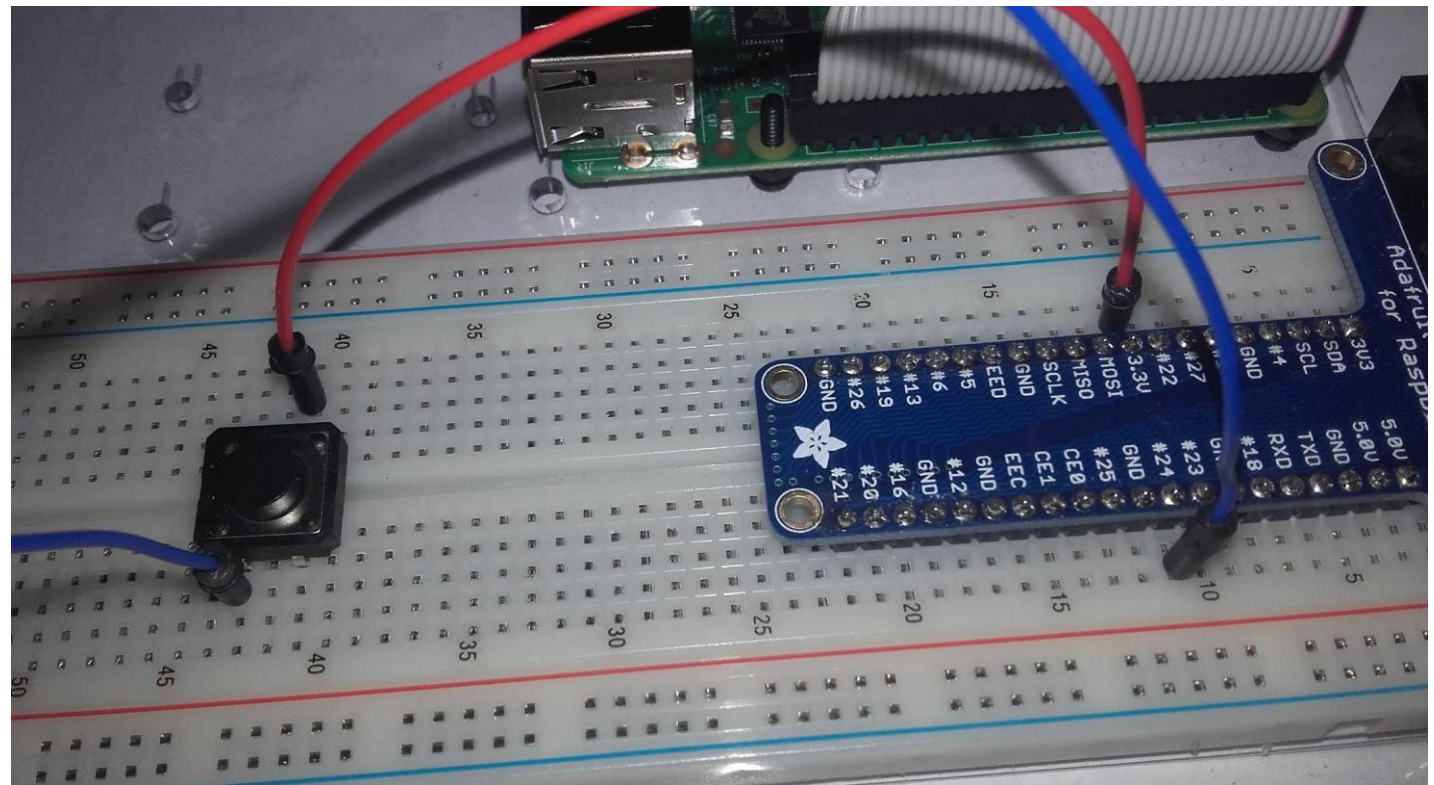
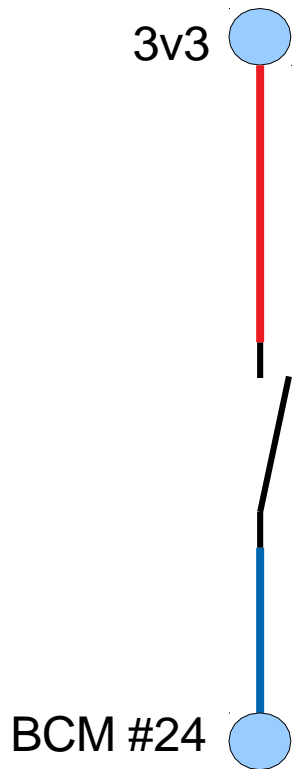
A terminal window titled 'pi@RaspEmmanuel: ~/PYTHON' showing the output of the script. It displays 15 lines of text: 'Port 24 est a l'etat BAS'. The window has a standard Linux terminal interface with a title bar and scrollbars.

```
pi@RaspEmmanuel: ~/PYTHON
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
```

Le bouton est enfoncé

## Le cas du second montage

Dans le second cas, BCM #24 remplace GND et on utilise la résistance interne en mode pull-down.



## Le cas du second montage

On exécute le même script :

```
#!/usr/bin/python3

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup (24, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

while True:

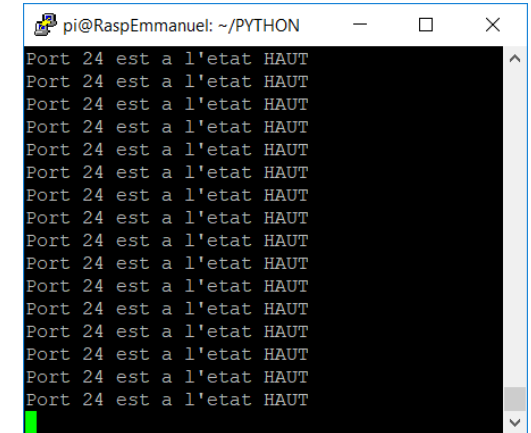
    if GPIO.input(24):

        print ("Port 24 est a l'etat HAUT")

    else:

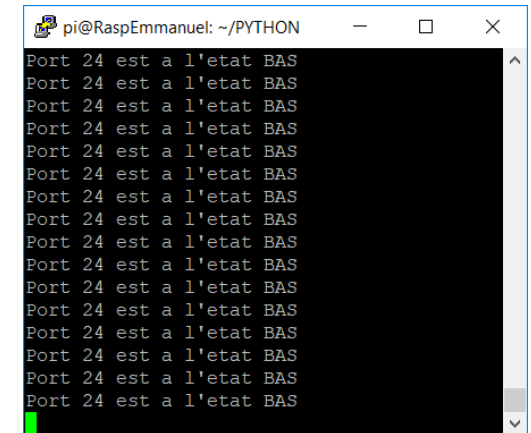
        print ("Port 24 est a l'etat BAS")

GPIO.cleanup()
```



```
pi@RaspEmmanuel: ~/PYTHON
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
Port 24 est a l'etat HAUT
```

Le bouton est **enfoncé**



```
pi@RaspEmmanuel: ~/PYTHON
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
Port 24 est a l'etat BAS
```

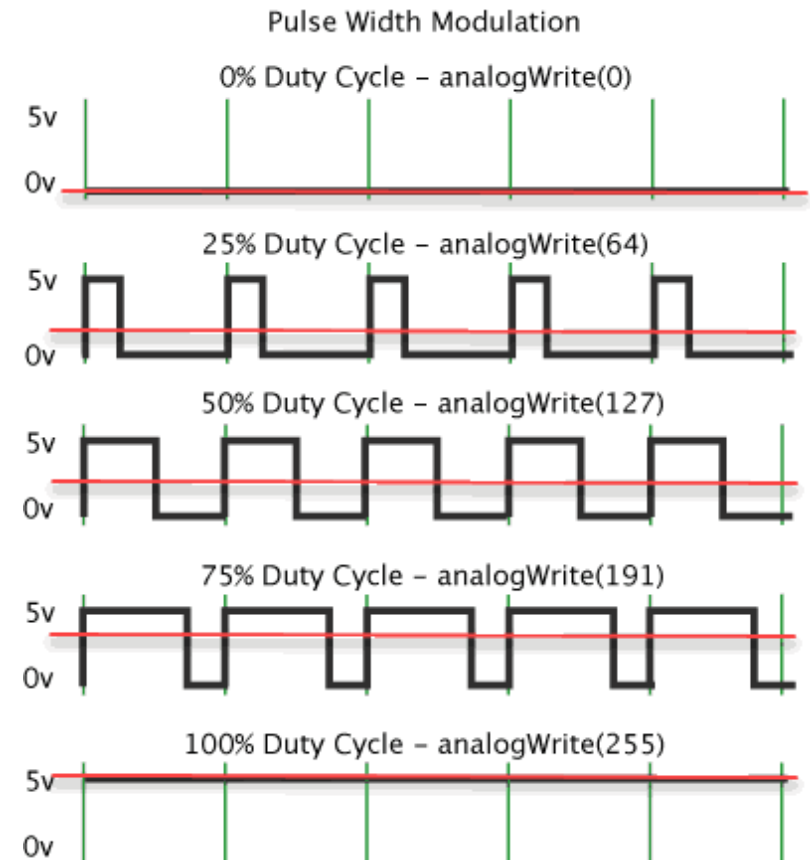
Le bouton est **relâché**

**Le PWM**

## Le principe

PWM est l'acronyme de « Pulse Width Modulation » qu'on peut traduire par « modulation de largeur d'impulsions ».

Cette technique consiste à générer un signal rectangulaire pour une période et un rapport cyclique (duty-cycle) donnés.

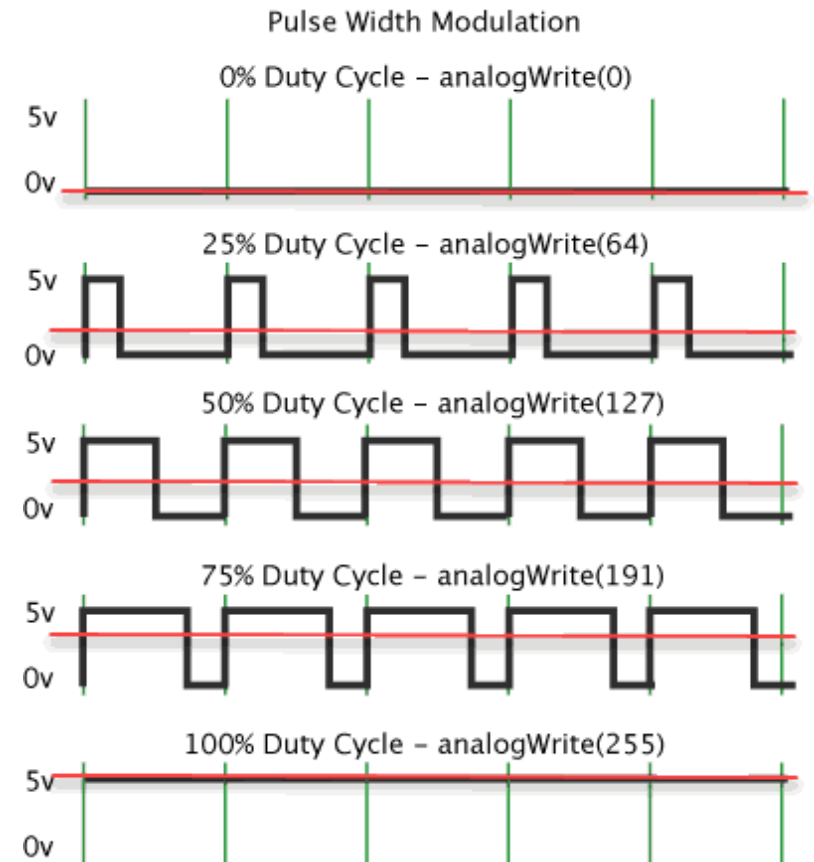


Voir <http://www.led-know-how.ch/fr/commande/variation/pwm-analogique-combinaison>

## Le principe

Cela permet de régler finement la puissance mis à disposition d'éléments électroniques de façon plus avantageuse que d'autres dispositifs comme des résistances variables :

- puissance consommée plus faible ;
- facilité de gestion de façon logicielles.





# Deux types de PWM sur le Raspberry Pi

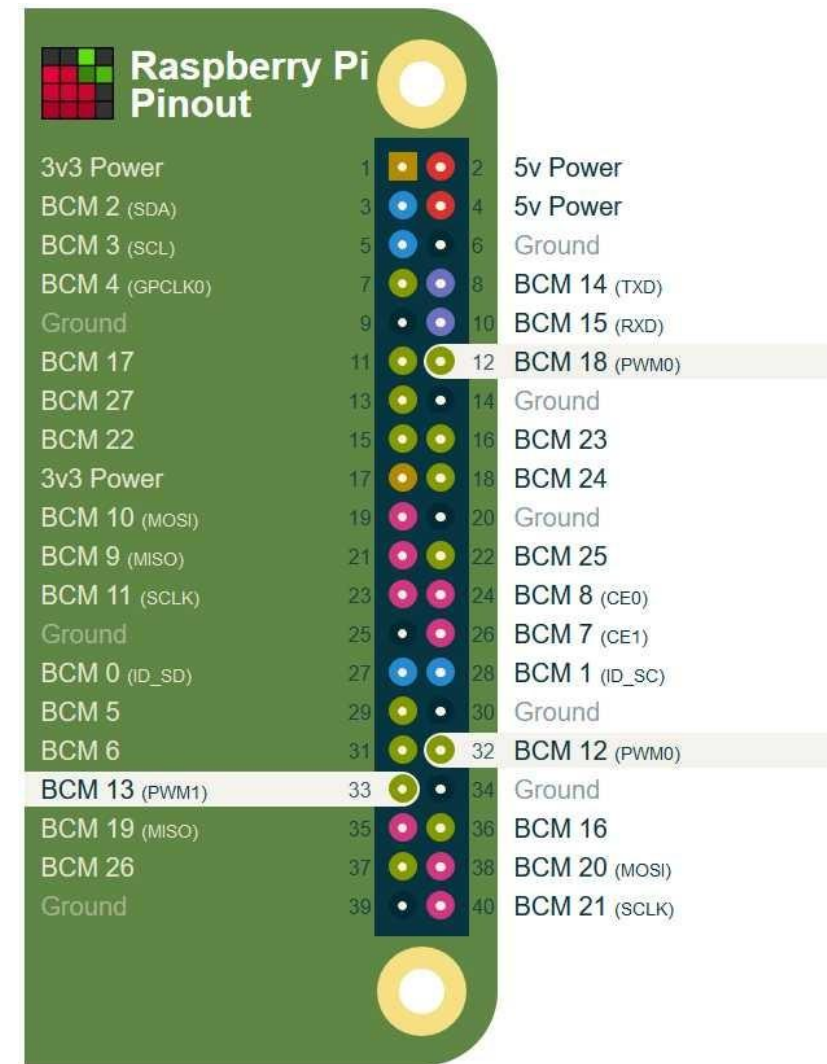
Au niveau du Raspberry Pi, on peut utiliser le PWM de deux manières :

- en utilisant les pattes du GPIO, qui peuvent être dédiées à cette fonction moyennant de instructions spécifiques ;
- en utilisant n'importe quelle sortie du GPIO, en utilisant d'autres instructions du module GPIO.

## Les pattes dédiées sur le GPIO

Le GPIO du Raspberry Pi dispose de 2 pins pouvant être affectés à cette tâche :

- le pin 12 correspond au PWM n°0 dans le mode Alt5 ;
- le pin 32 correspond au PWM n°0 dans le mode Alt0 ;
- le pin 33 correspond au PWM n°1 dans le mode Alt0 ;



Voir [https://pinout.xyz/pinout/pin32\\_gpio12#](https://pinout.xyz/pinout/pin32_gpio12#)

## Passer en mode Alt0 ou Alt5

Pour utiliser ces 3 pattes dans un mode de fonctionnement PWM, il est nécessaire de passer le GPIO en Alt0 ou Alt5. Cependant, le passage du GPIO dans un mode Alt ne semble pas encore prise en charge par la bibliothèque GPIO.

Cette solution n'est donc pas utilisable pour le moment bien qu'elle aurait permis une gestion plus matériel (au niveau du microprocesseur) de cette fonctionnalité PWM.

Il faut se rabattre sur des bibliothèques en langage C comme WiringPi.

Voir <https://raspberrypi.stackexchange.com/questions/29366/change-gpio-pin-mode-to-alt-with-python-rpi-gpio>  
<https://www.raspberrypi.org/forums/viewtopic.php?t=39138>

# Les nouvelles instructions

Pour utiliser la seconde solution, il faut utiliser les nouvelles instructions ci-dessous :

- `p = GPIO.PWM (pin, fréquence)`

Cette instruction permet de créer un objet qui va prendre en charge la génération d'un signal carré dont la fréquence et le pin de sortie sont mentionnés en paramètre

- `p.start (rapport_cyclique)`

Cette instruction lance la génération du signal carré avec un rapport cyclique compris entre 0.0 et 100.0

# Les nouvelles instructions

- `p.ChangeFrequency (frequence)`

Cette instruction permet de changer la fréquence du signal carré en cours de génération

- `p.ChangeDutyCycle (rapport_cyclique)`

Cette instruction permet de modifier le rapport cyclique du signal carré en cours de génération

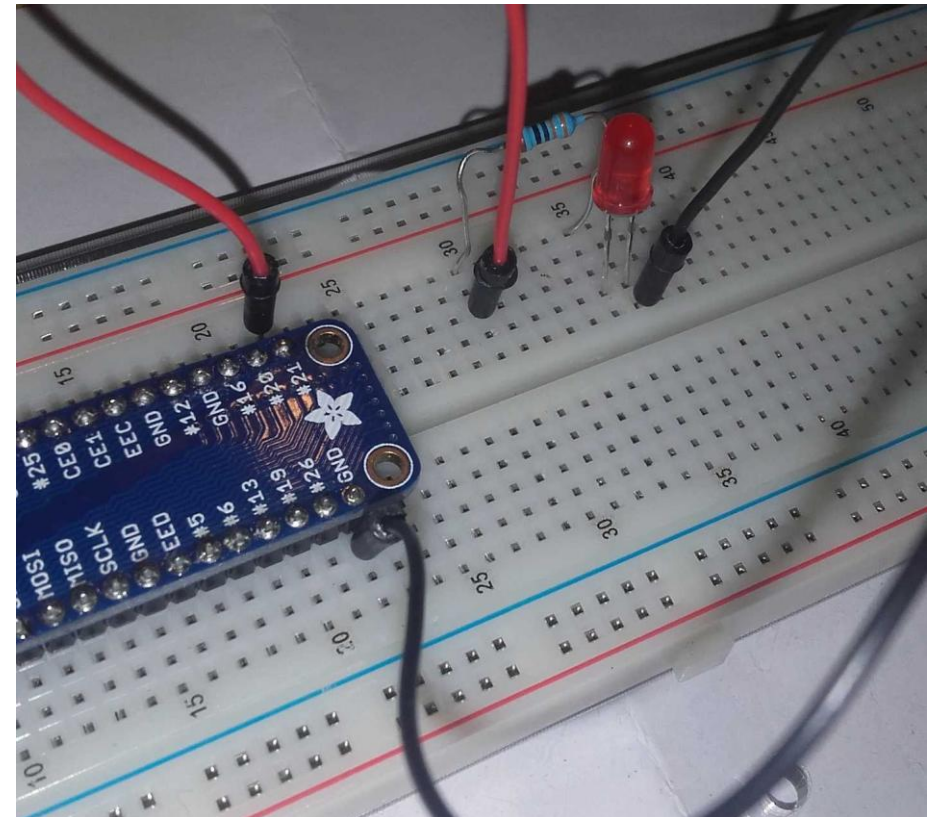
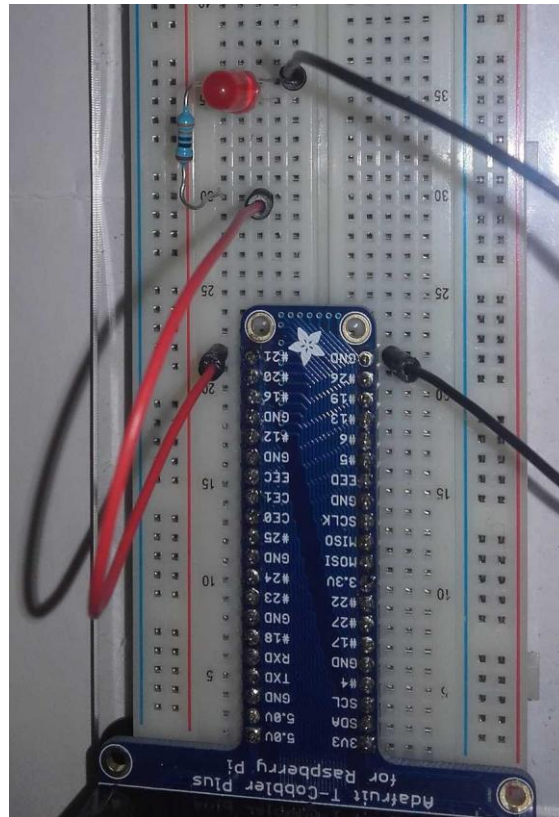
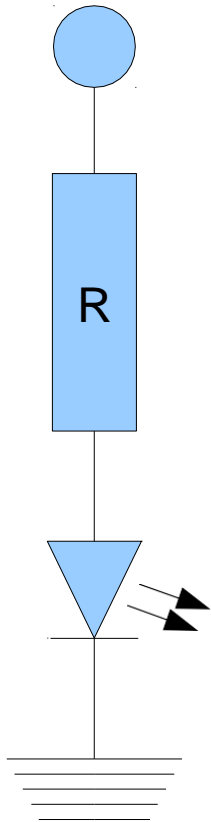
- `p.stop()`

Cette fonction est utilisée pour stopper la génération du signal carré.

Voir <http://deussyss.developpez.com/tutoriels/RaspberryPi/PythonEtLeGpio/>

## Exemple

On reprend le montage réalisé dans la partie x avec pour objectif de moduler la puissance envoyé vers la led



# Exemple

On peut utiliser ce premier script pour constater que la led clignote avec une fréquence et un rapport cyclique différents à chaque frappe de touche.

```
#!/usr/bin/python3

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setup (21, GPIO.OUT, initial=GPIO.LOW)

touche = input ('Frappez une touche')
p = GPIO.PWM (21, 1)
p.start (50)

touche = input ('Frappez une touche')
p.ChangeFrequency(2)

touche = input ('Frappez une touche')
p.ChangeDutyCycle (100)

touche = input ('Frappez une touche')
p.stop()
GPIO.output (21, GPIO.LOW)

GPIO.cleanup ()
```

# Exemple

Le script, fourni à la page suivante, est une évolution du script précédent. Il utilise des fréquences plus élevées de l'ordre de 100 Hz où la led commute très rapidement.

Le rapport cyclique permet alors de moduler la puissance fournie à la led donc sur la quantité de lumière qu'elle émet.

Voir <https://www.digikey.fr/fr/articles/techzone/2016/oct/how-to-dim-an-led-without-compromising-light-quality>



## Exemple

```
#!/usr/bin/python3

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup (21, GPIO.OUT, initial=GPIO.LOW)

touche = input ('Frappez une touche')

frequence = 101
rapport    =    0

p          = GPIO.PWM (21, frequence)
p.start (rapport)

print ("freq=", frequence, " et rapport=", rapport)

print ("On fait evoluer le rapport cyclique")

for i in range (10) :

    touche = input ('Frappez une touche')
    rapport += 10
    p.ChangeDutyCycle (rapport)
    print ("freq=", frequence, " et rapport=", rapport)
```

# Exemple

```
print ("On va faire evoluer la frequence avec un rapport cyclique a 50")

touche = input ('Frappez une touche')
rapport = 50
p.ChangeDutyCycle (rapport)
print ("freq=", frequence, " et rapport=", rapport)

for i in range (10) :

    touche = input ('Frappez une touche')
    frequence -= 10
    p.ChangeFrequency(frequence)
    print ("freq=", frequence, " et rapport=", rapport)

touche = input ('Frappez une touche pour arreter')

p.stop()
GPIO.output (21, GPIO.LOW)

GPIO.cleanup ()
```