



# MicroPython - Entrées, Sorties (GPIO)



[Mise à jour le : 22/5/2023] **En cours de rédaction**

- **Ressources**

- [MicroPython.org](https://micropython.org)
- [MicroPython documentation](#)
- [IDE Thonny](#)

- **Lectures connexes**

- [MicroPython - Les modules Raspberry Pi Pico et Pico W](#)
- [ESP32/ESP8266 Digital Inputs and Digital Outputs with MicroPython](#)
- [MicroPython with ESP32 and ESP8266: Interacting with GPIOs](#)
- [ESP32/ESP8266 PWM with MicroPython - Dim LED](#)
- [ESP32/ESP8266 Analog Readings with MicroPython](#)
- [MicroPython: Interrupts with ESP32 and ESP8266](#)

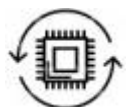
## 1. Généralités

« Dans un système à base de **microcontrôleur**, on appelle **entrées-sorties** les échanges d'informations entre le processeur et les périphériques qui lui sont associés. De la sorte, le système peut réagir à des modifications de son environnement, voire le contrôler. Elles sont parfois désignées par l'acronyme **I/O**, issu de l'anglais **Input/Output** ou encore **E/S** pour **entrées/sorties**. » [Wikipédia](#)

Pour éviter de faire référence à des valeurs électriques (tension ou intensité), on définit souvent l'état d'un signal numérique en utilisant la logique booléenne.

- **true** (« 1 » logique) correspondra par exemple à 5V ou 3,3V

- **false** (« 0 » logique) correspondra à 0V.



Un **microcontrôleur** dispose de broches pouvant être contrôlées par un logiciel. Elles peuvent se comporter comme des entrées ou des sorties, d'où le nom "entrée / sortie à usage général", ou **GPIO** (**G**eneral **P**urpose **I**nput **O**utput. Le nombre de broches d'un microcontrôleur étant limité, il est fréquent d'avoir plusieurs fonctionnalités sur une même broche.

Les exemples de code de cette page ont été testés sur une [ESP32 Feather Huzzah](#) et sur une [Raspberry Pi Pico](#).

## 2. Entrées, sorties numériques

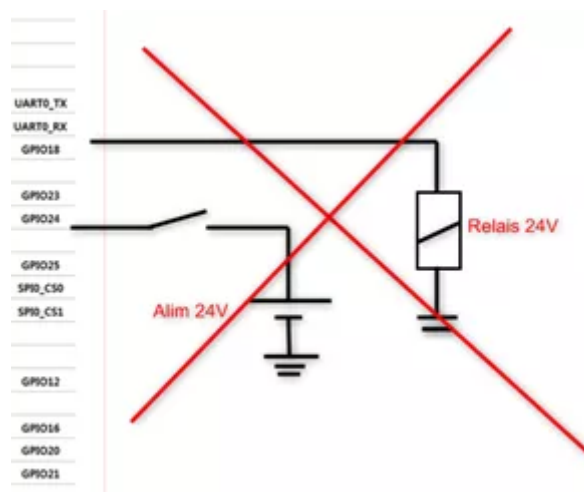


### 2.1 Sortie numérique

- **Matériels** : [ESP32 Feather Huzzah](#) ou [Raspberry Pi Pico](#)

#### 2.1.1 Présentation

Une sortie numérique est fragile. Ne **JAMAIS** la relier à un générateur  
Une sortie numérique délivre **très peu de puissance** (quelques centaines de mW). Il n'est donc pas possible de la relier directement à un actionneur (moteur). Il est nécessaire de placer une interface de puissance (hacheur, relais) entre elle et l'actionneur à commander.



### 2.1.2 Exemples de code

- [RPi Pico](#)
- [ESP32](#)
- **Ressources** sur Micropython.org.
  - [module machine](#) | [module time](#) | [class Pin – control I/O pins](#)
  - [Quick reference for the RP2, Pins and GPIO](#)

Exemple de code pour un **Raspberry Pi Pico**

\*.py

```
# -----  
# -----  
# Faire clignoter la led de la carte Raspberry Pi Pico  
# Date : 22/5/2023  
# Matériels : Raspberry Pi Pico, Shield Grove  
# IDE : Thonny  
# Fichiers : blink.py  
# -----  
# -----  
  
from machine import Pin  
import time  
  
# Configuration de la broche associée à la led de la carte  
led = Pin('LED',Pin.OUT) # 'LED' <=> 25 (GPIO25)  
  
while (True):  
    led.on()  
    time.sleep(0.5) # Attente 0,5s  
    led.off()  
    time.sleep(0.5)
```

- **Ressources** sur Micropython.org.
  - [module machine](#) | [module time](#) | [class Pin – control I/O pins](#)
  - [Quick reference for the ESP32, Pins and GPIO](#).

Exemple de code pour un **ESP32 Feather Huzzah**

\*.py

```
# Faire clignoter la led de la carte ESP32  
# Bibliothèques à installer  
from machine import Pin  
import time  
  
# Configuration de la broche associée à la led de la carte
```

```
led = Pin(13, Pin.OUT)

while (True):
    led.on()
    time.sleep(1) # Attente 1s
    led.off()
    time.sleep(1)
```



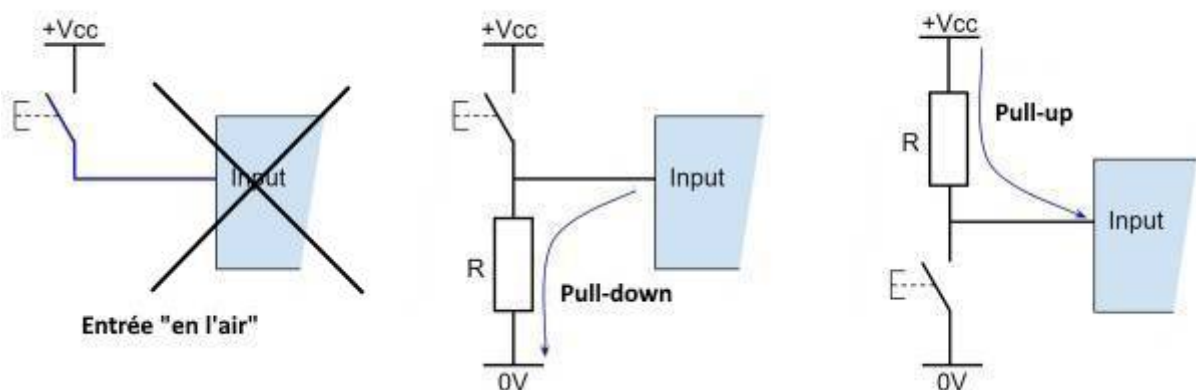
## 2.2 Entrée numérique

- **Matériels** : [ESP32 Feather Huzzah](#) ou [Raspberry Pi Pico](#), [Digilent Pmod BTN: 4 User Pushbuttons](#) [[Schéma](#)]

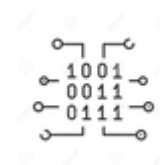
### 2.2.1 Présentation

Les entrées numériques sont **fragiles**. Elles ne supportent ni les **décharges électrostatiques** ni les **surtensions**. Il ne faut ni les toucher ni leur appliquer une tension supérieure à 5V ou inférieure à 0V.  
Une entrée numérique utilisée dans un programme **ne doit pas être laissée "en l'air"** (non connectée) car elle prendra alors un état logique aléatoirement et le comportement du programme deviendra imprévisible.

#### Résistance de rappel / tirage / Pull-up / Pull-down



Les microcontrôleurs disposent de **résistances de rappel internes** pouvant être connectées par le logiciel.



### 2.2.2 Exemples de code

- [RPi Pico](#)
- [ESP32](#)
- **Ressources** sur Micropython.org.
  - [module machine](#) | [module time](#) | [class Pin - control I/O pins](#)
  - [Quick reference for the RP2, Pins and GPIO](#)

Exemple de code pour un **Raspberry Pi Pico**

\*.py

```
# Configuration (en entrée) des broches connectées à deux boutons-
poussoirs
# Bibliothèques à installer
from machine import Pin

# Configuration (en entrée) des broches connectées à deux boutons-
poussoirs
button_min = Pin(20, Pin.IN)
button_hr = Pin(21, Pin.IN)
...
```

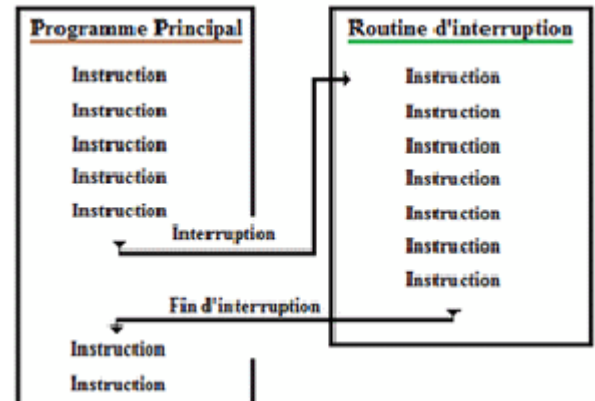
- **Ressources** sur Micropython.org.
  - [module machine](#) | [module time](#) | [class Pin - control I/O pins](#)
  - [Quick reference for the ESP32, Pins and GPIO](#)

Exemple de code pour un **ESP32 Feather Huzzah**

\*.py

```
# Configuration (en entrée) des broches connectées à deux boutons-
poussoirs
button_min = Pin(25, Pin.IN)
button_hr = Pin(26, Pin.IN)
...
```

## 2.3 Interruption



- **Matériel** : [ESP32 Feather Huzzah](#) ou [Raspberry Pi Pico](#), [Digilent Pmod BTN: 4 User Pushbuttons](#) [Schéma]

### 2.3.1 Présentation

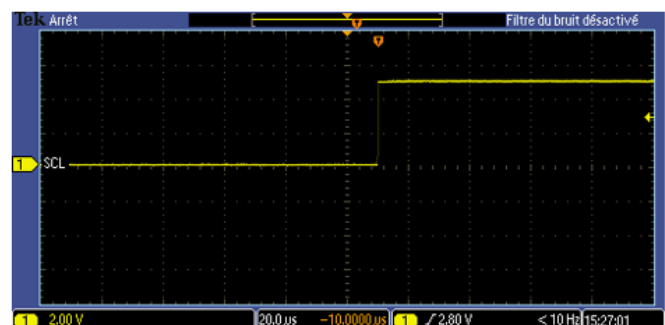
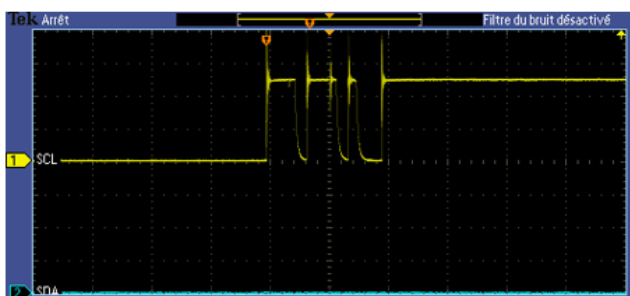
Une **interruption** est un **arrêt temporaire** de l'exécution normale d'un programme par le processeur afin d'exécuter un autre programme (appelé **service d'interruption**).

L'interruption est provoquée par une cause externe (action sur un bouton-poussoir, mesure réalisée par un capteur, horloge temps réel, etc.).

On utilise les interruptions afin de permettre des **communications non bloquantes** avec des périphériques externes.

Une interruption tient compte de l'état logique présent sur une broche. Couramment, on la déclenchera sur **le front montant, le front descendant, ou chacun des fronts** d'un signal logique.

Une interruption sera reconnue si le signal présente des fronts "propres". Il faudra donc s'assurer de la qualité du signal. Les figures ci-dessous représentent un signal transmis à la fermeture du contact d'un anémomètre. Le signal de gauche n'est pas utilisable à cause du rebondissement du contact. En effet, il contient quatre fronts montants au lieu d'un seul comme dans le cas du signal de droite.

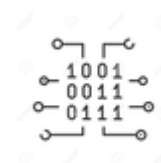


### 2.3.2 Configuration

La configuration en entrée de la broche destinée à recevoir un évènement est identique à celle du paragraphe précédent.

### 2.3.3 Evènement et gestionnaire d'évènement

Un évènement est attaché à un gestionnaire (service d'interruption) .



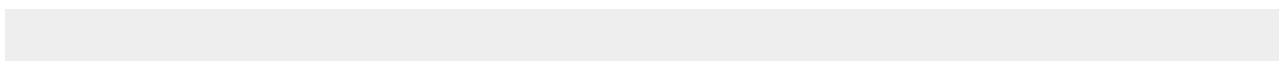
### 2.3.4 Exemples de code

- [RPi Pico](#)
- [ESP32](#)

#### A faire

Exemple de code pour un **Raspberry Pi Pico**

\*.py



- **Ressource** : [Quick reference for ESP32, GPIO Pins, External interrupts](#) sur Micropython.org.

Exemple de code pour un **ESP32 Feather Huzzah**

\*.py

```
# Code partiel du programme HORLOGE

# Réglage de l'heure à la mise sous tension
time_offset=12*3600+0*60+0 # hh+mm+ss

# Routines de service d'interruption (ISR)
def handle_interrupt_min(pin):
    global time_offset
    time_offset+=60
    time.sleep(.2)
```

```
def handle_interrupt_hr(pin):
    global time_offset
    time_offset+=3600
    time.sleep(.2)

# Réglage des minutes
# Ajout de 60s à l'heure initiale
button_min = Pin(25, Pin.IN)
# Gestionnaire d'interruption
button_min.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt_min)

# Réglage des heures
# Ajout de 3600s à l'heure initiale
button_hr = Pin(26, Pin.IN)
# Gestionnaire d'interruption
button_hr.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt_hr)
{{ :python:micropython:materiel:thonny.png?nolink&70| }}
```

**Télécharger** le projet MICROPYTHON\_ESP32\_HORLOGE pour Thonny.



A voir : la vidéo de démonstration sur [Youtube](#)



### 3. Entrées analogiques

- **Ressource**
  - [Quick reference for the RP2, Pins and GPIO](#) sur Micropython.org., potentiomètre 10kOhm.

#### 3.1 Présentation

- **Ressource**
  - [Un signal analogique : petits rappels](#) sur le site Zeste de savoir.





### 3.2 Exemples de code

- [RPI Pico](#)
- [ESP32](#)
- **Ressource**
  - [ADC \(analog to digital conversion\)](#) sur [Micropython.org](#).

Exemple de code pour un **Raspberry Pi Pico**

\*.py

```
# -----
# -----
# Lecture et affichage dans la console de la tension issue d'un
# potentiomètre
# Date : 22/5/2023
# Matériels : Raspberry Pi Pico, Shield Grove, pot. 10k
# ADC accessibles sur le shield Grove pour RP2 :
# Connecteur: ADC      : GPIO
#      A0 : ADC0      : 26
#      A1 : ADC0,ADC1: 26,27
#      A2 : ADC1,ADC2: 27,28
# IDE : Thonny
# -----
# -----
from machine import ADC, Pin
import time

# Le potentiomètre 10k0hm est connecté à l'entrée analogique A0 du
# shield.
# Attention : La tension doit être comprise entre 0 - 3,3V (3,6V max !)
# sur une entrée analogique.
# Configuration
pot = ADC(Pin(26))

while (True):
    val=pot.read_u16() # lecture de l'ADC
    U = val*3.3/65535 # Calcul de la tension
    print("%.2f" % U) # Affichage dans la console (formaté à 2
# décimales)
    time.sleep(1)
```

- **Ressource**

- [ADC \(analog to digital conversion\)](#) sur Micropython.org.

Exemple de code pour un **ESP32 Feather Huzzah**

`*.py`

```
# ADC accessibles en Python sur la carte ESP32 Feather Huzzah :
# ADC:GPIO
# A2 : 34
# A3 : 39
# A4 : 36
# A7 : 32
# A9 : 33

from machine import ADC, Pin

# Le potentiomètre 10k0hm est connecté à l'entrée analogique A2 de
l'ESP32.
# Configuration
adc = ADC(Pin(34))
# Sur une entrée analogique, la tension doit
# être comprise entre 0 - 3,3V (3,6V max !)
adc.atten(ADC.ATTN_11DB) # voir doc
# Mesure
value = adc.read()

print(value) # affichage dans la console
```

From:

<http://webge.fr/dokuwiki/> - **WEBGE Wikis**

Permanent link:

<http://webge.fr/dokuwiki/doku.php?id=python:micropython:es&rev=1684767142>

Last update: **2023/05/22 16:52**

