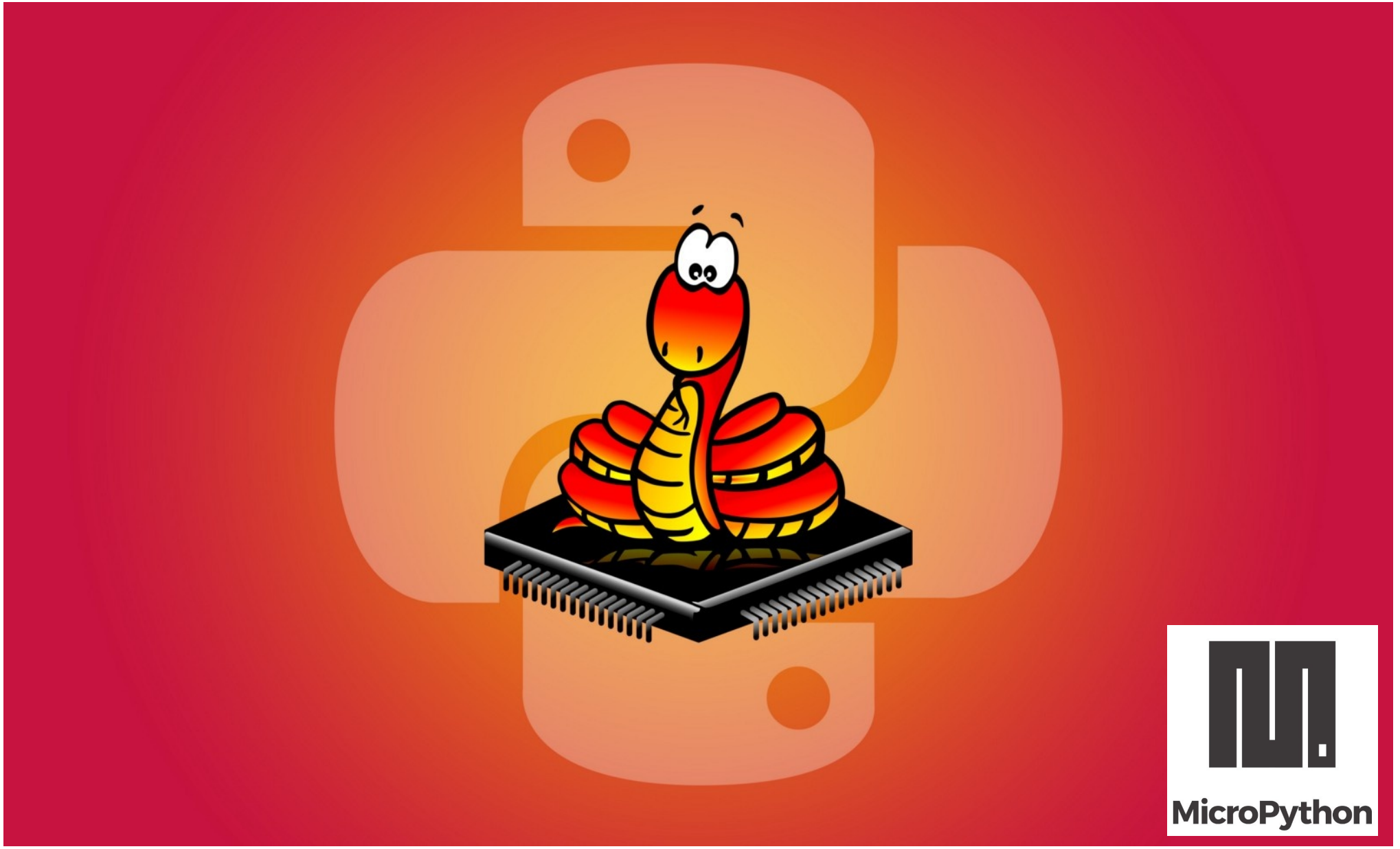
# MicroPython



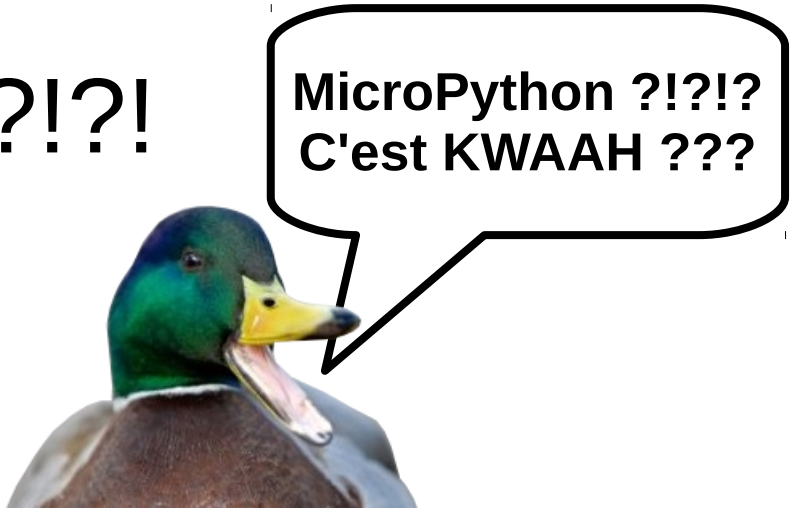
# Qui sommes nous ?

- `wiki.mchobby.be` **wiki open-source** sur Arduino, Raspberry Pi, MicroPython
- Partager le savoir au plus grand nombre
- WebShop `shop.mchobby.be` avec des **produits documentés**

# Qui suis-je ?

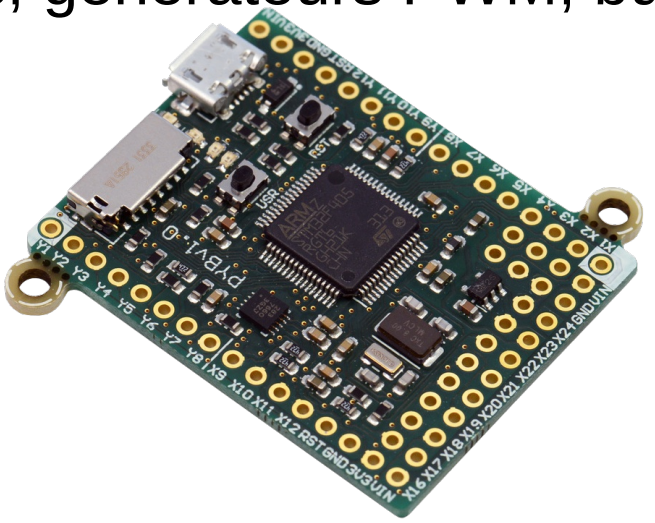
- Dominique Meurisse
- Développeur, Architecte technique
- Auteur chez ENI
  *Python, Raspberry-Pi et Flask.*
- « Évangéliste » Python & MicroPython

# MicroPython ?!?!




**Python 3 pour Microcontrôleur**

- Implémentation de Python 3 légère, efficace.
  Optimisé pour l'exécution sur Microcontrôleur (16K Ram, 256 Flask)
- Implémente un sous-ensemble des bibliothèques standards de Python
- S'exécute en BareMetal et offre donc un accès à l'architecture sous-jacente (les broches, les timers, générateurs PWM, bus matériel I2C / SPI / UART, RTC,
  le **système de fichier en Flash**)
- MicroPython **Pyboard**.
  Première implémentation
  matérielle de MicroPython




PC

Environnement développement

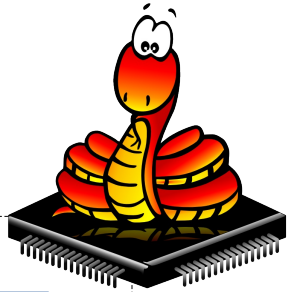
Code

Compilateur → binaire

Micro Contrôleur

Flash

Microcontrôleur « *cpu* »



PC

Editeur de texte

USB Mass Storage

Micro Contrôleur

Flash

boot.py

main.py

Python script

Byte code

Flash

MicroPython Firmware

Microcontrôleur « *cpu* »

# Carte PyBoard

Micro Python PyBoard = 3.3v

Connecteur MicroSD.
Etendez le système de fichier

4 LEDs (P2 à P5)

Connecteur micro USB.
* Alimentation
* USB-Serial - vu comme un port série via USB
* HID - vu comme clavier souris
* MSD - vu comme un périphérique de stockage USB

VIn: tension d'entrée entre 3.6 et 10v

Sortie du régulateur.
3.3V 300mA max

Bouton utilisateur
Broche X17

Bouton de réinitialisation (Reset)

Microcontroleur STM32F405RG
CPU Cortex-M4, 168 Mhz
1 Mb Flash, 192 Kb RAM
Horloge temps réel

Accéléromètre

30x broches GPIO.
Communication:
* 2 bus SPI
* 2 bus CAN
* 2 bus I2C
* 5 USART (port série)
14x entrées analogiques
résolution 12 bits
2x sorties analogiques

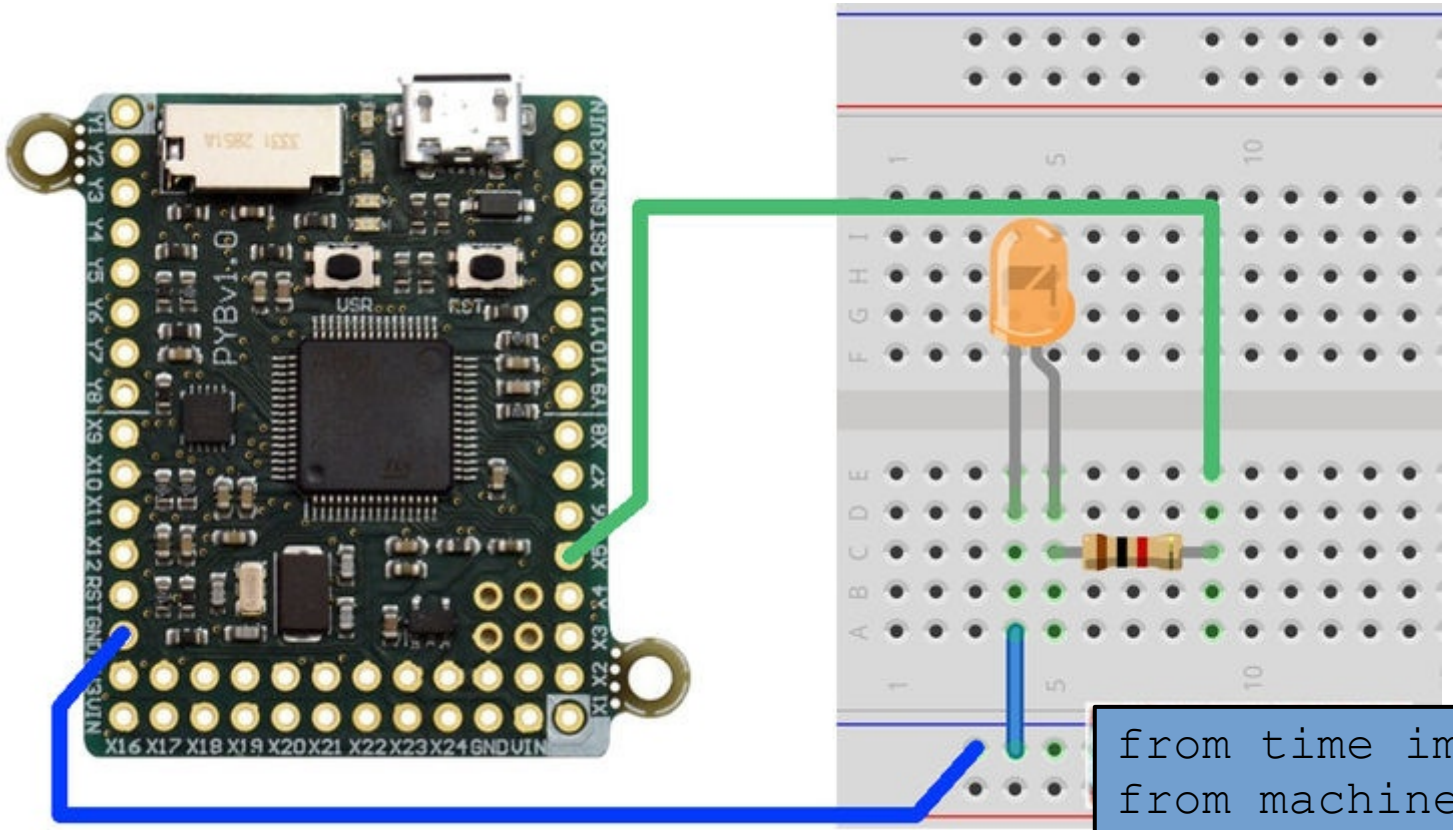
Branchement de Servo-moteur

VIN est utilisé comme puissance d'alimentation.

Régulateur 3.3v
Low Drop Out

Source: MicroPython.org
Composition: MCHobby.be

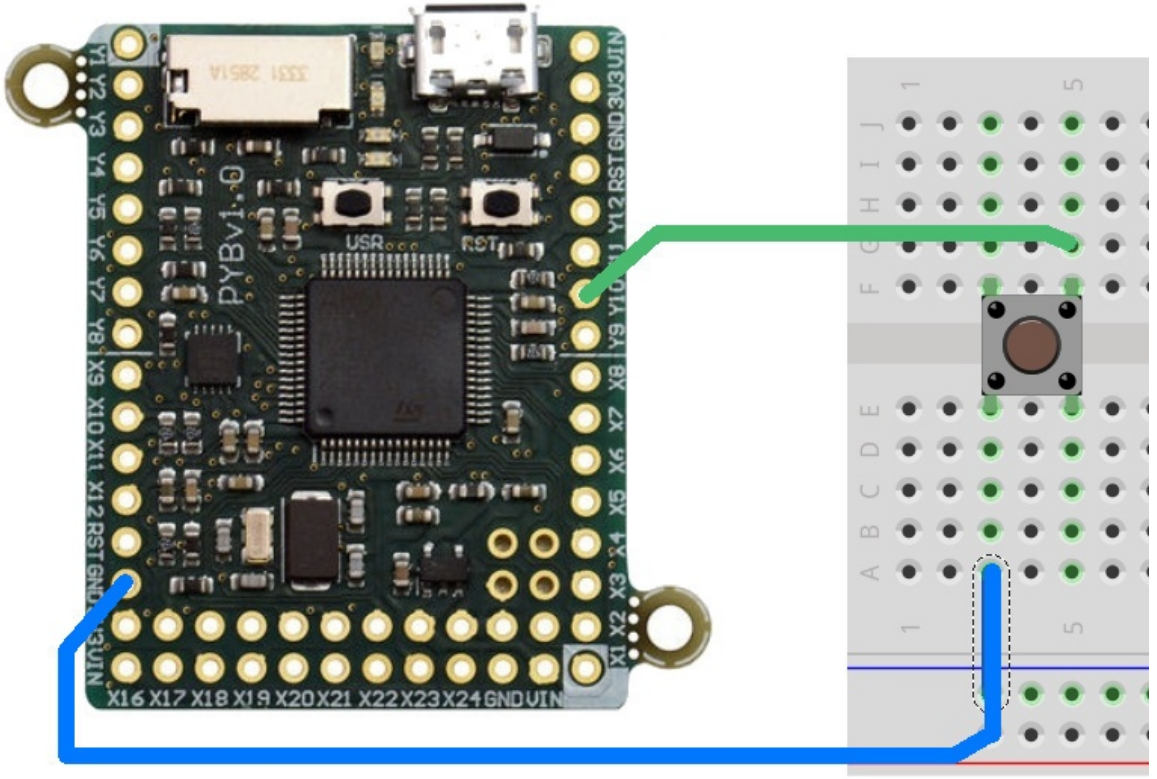
# Commande d'une LED



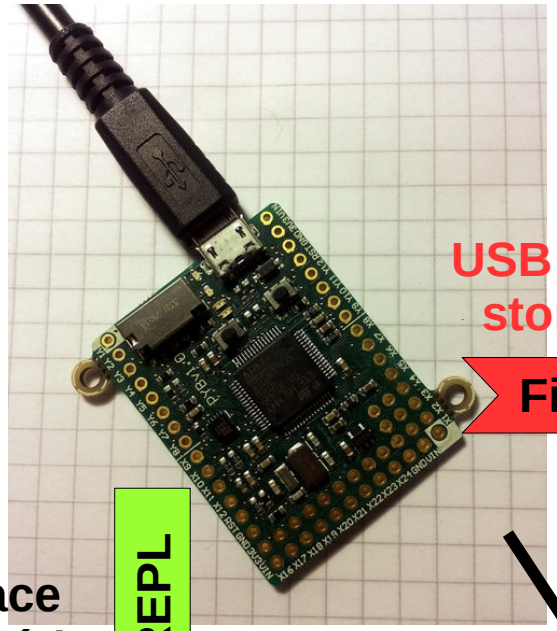

```
from time import sleep
from machine import Pin
led = Pin( 'X5', Pin.OUT)
while True :
    led.value( 1 ) # allumé
    sleep( 1 ) # 1 sec
    led.value( 0 ) # éteint
    sleep( 1 )
```

# Entrée bouton

```
from time import sleep
from machine import Pin
btn = Pin( 'Y10', Pin.IN, Pin.PULL_UP)
while True :
    print( pin.value() ) # 0 ou 1
    print( 'Pressé' if pin.value()==0 else 'ouvert' )
    led.value( 0 ) # éteint
    sleep( 1 )
```

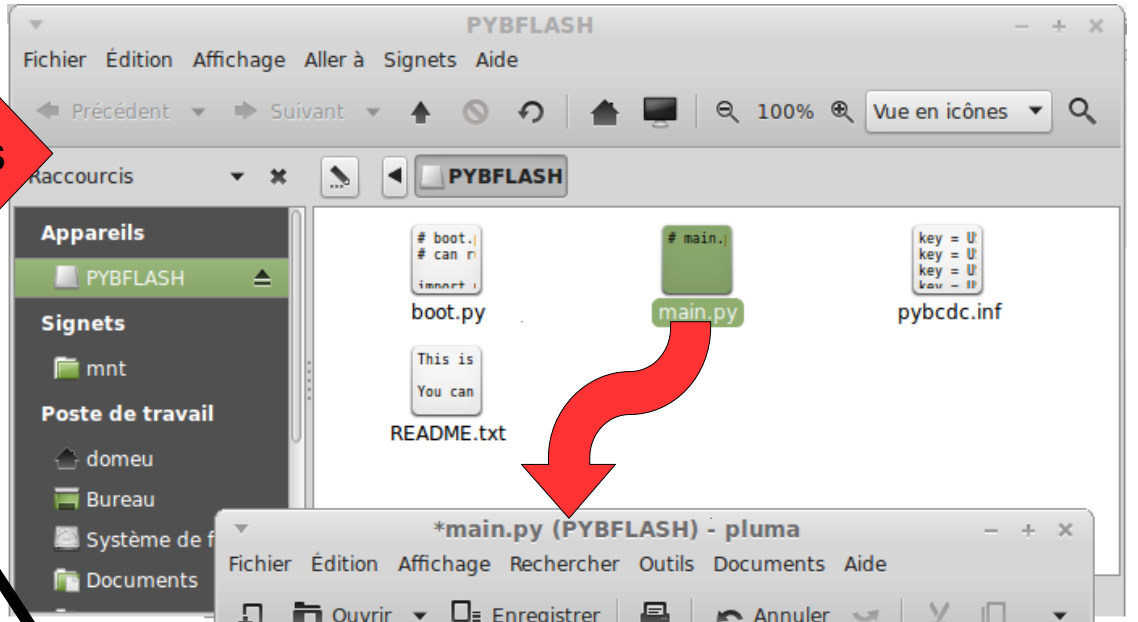


# Connecté en USB
# → 2 méthodes d'accès



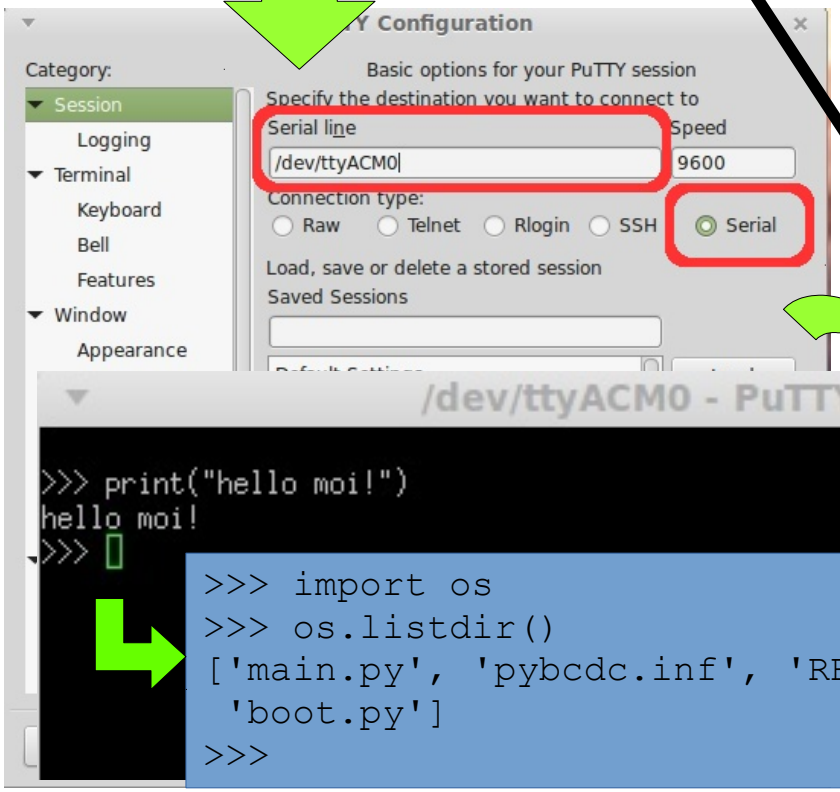
**USB Mass storage**

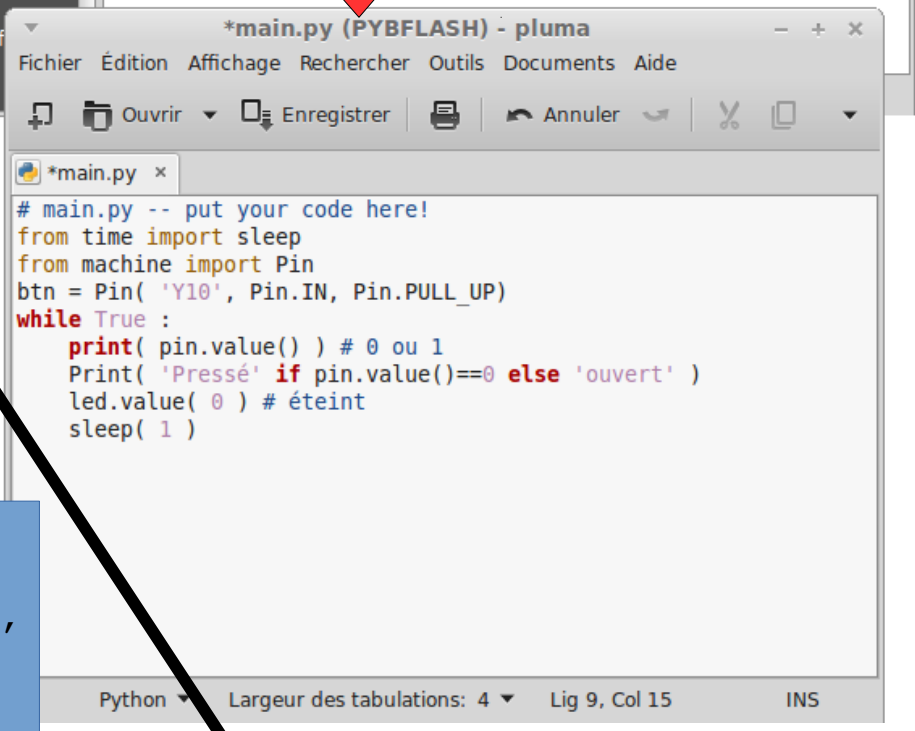
**Fichiers**




**Interface USB Série**

**REPL**



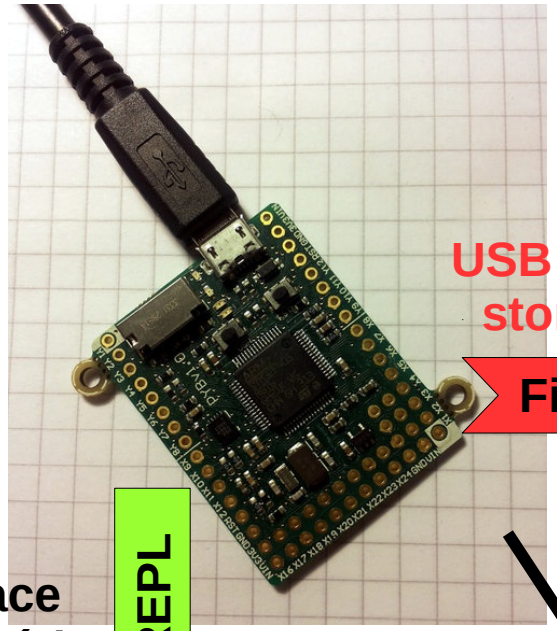







**REPL :** Read-Evaluate-Print-Loop (boucle de lecture-évalutation-affichage)

# Des outils pratiques



USB Mass storage

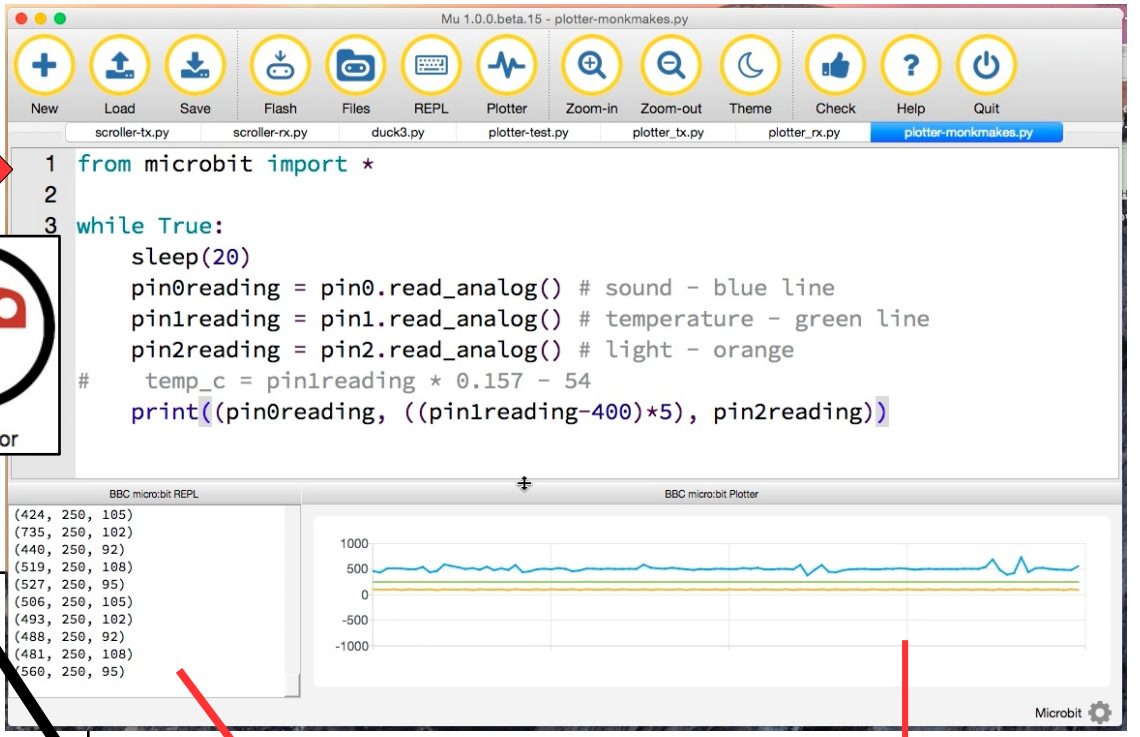
Fichiers

Interface USB Série

REPL

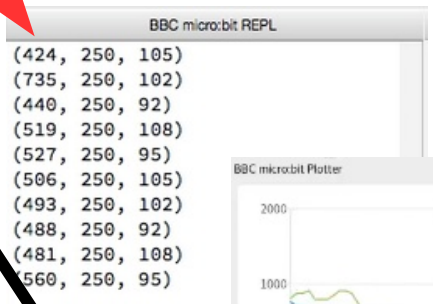
RShell

mu-editor




```
Welcome to rshell. Use Control-D to exit.
> ls -l /flash
   529 May 21 17:34 README.txt
   286 May 21 17:34 boot.py
    34 May 21 17:34 main.py
  2436 May 21 17:34 pybcdc.inf
> cp hello.py /flash
> ls -l /flash
   529 May 21 17:34 README.txt
    21 May 21 17:35 hello.py
   ...
> cat /flash/hello.py
print('Hello World')
> repl
Micro Python v1.4.3-28-ga3a14b9 on 2015-05-21; PYBv1.0
with STM32F405RG
>>>
>>> import hello
Hello World
>>>
```

Session REPL depuis RShell

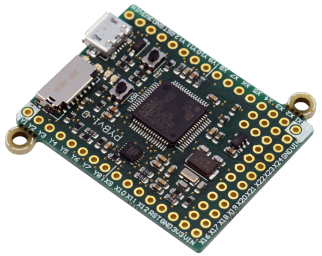




**REPL :** Read-Evaluate-Print-Loop (boucle de lecture-évalutation-affichage)

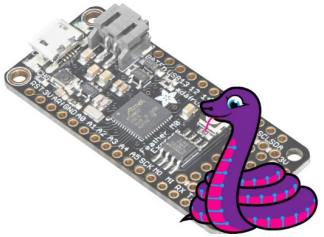
# Les cartes MicroPython

**STMicroelectronic**
STM32F405
ARM Cortex 32bits



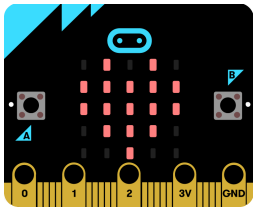
MicroPython

**Microchip**
ATSAMD21
Cortex M0 32bits
(Arduino Zero)



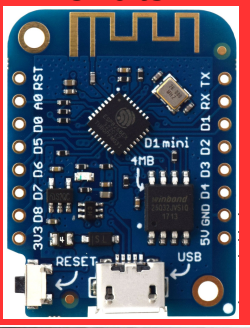
adafruit INDUSTRIES

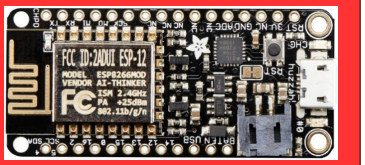
**Nordic Semiconductor**
nRF51822
Cortex M0 32bits
(Arduino Zero)



micro:bit

**ESP8266**
Xtensa LX106
32bits



MicroPython

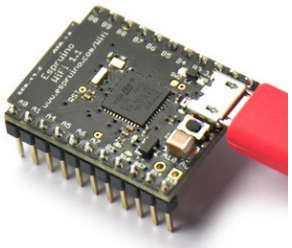
**ESP32**
WROOM32
32bits

MicroPython

**WiPy**
ESP32


pycom go invent

**STMicroelectronic**
STM32



Espruino

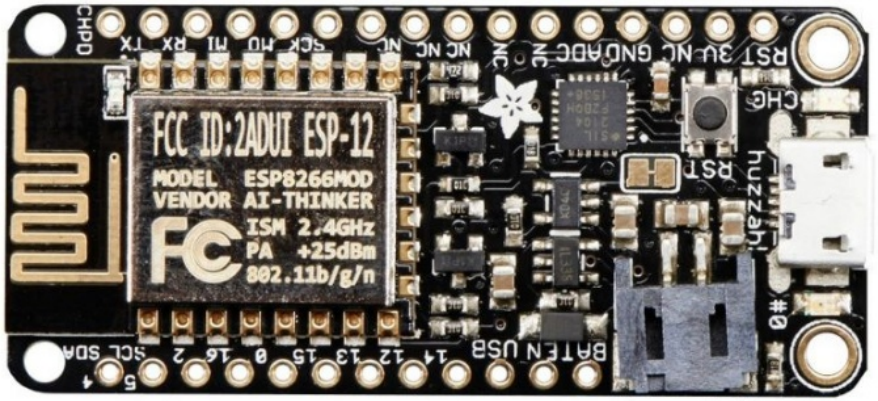
Info

Plus d'information sur ces produits disponible sur

https://shop.mchobby.be

# Pourquoi MicroPython sur ESP8266 ?

# MicroPython sur ESP8266




**Ressources**

- ESP8266 @ 80MHz
- 4 Mb de mémoire FLASH
- 96 KiB RAM ⚠
- Logique 3.3V
- USB-Série CP2104
- Pas de USB Mass Storage
- 9 broches GPIO disponibles
- 1 entrée analogique (1V max)

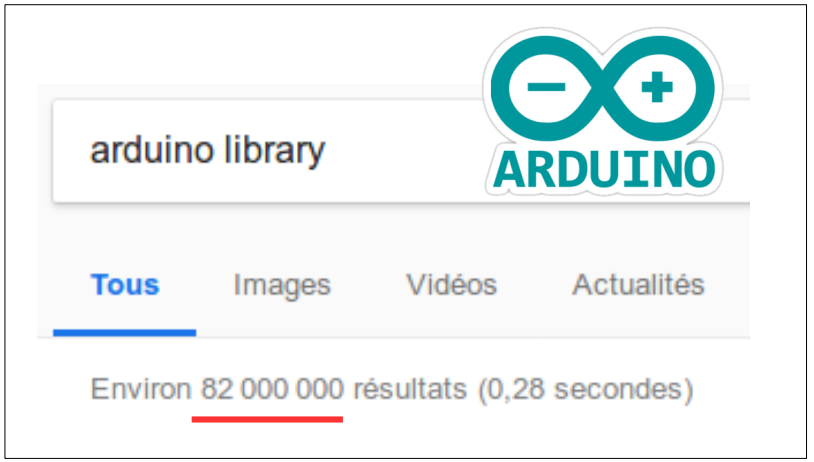
**Les points positifs**

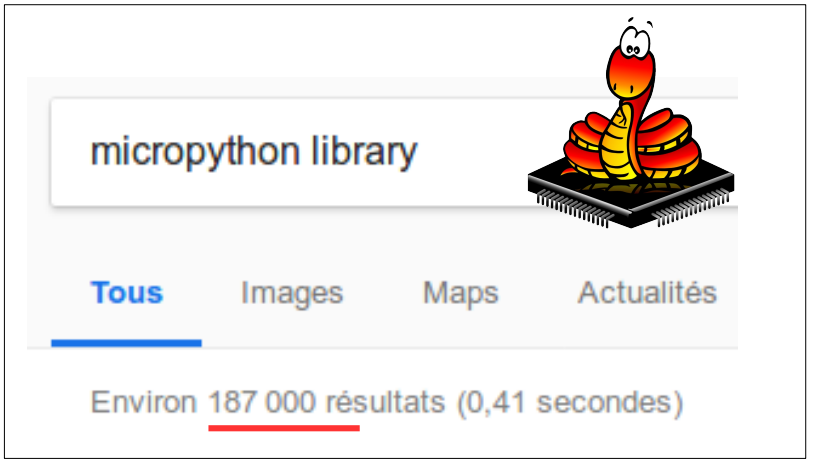
- Système de fichiers en Flash
- REPL ← RShell
- Interface WiFi (STA / AP)
- Socket
- MQTT
- WebREPL
- Possibilité Telnet / FTP (communauté)
- I2C Bus (bit-banging)

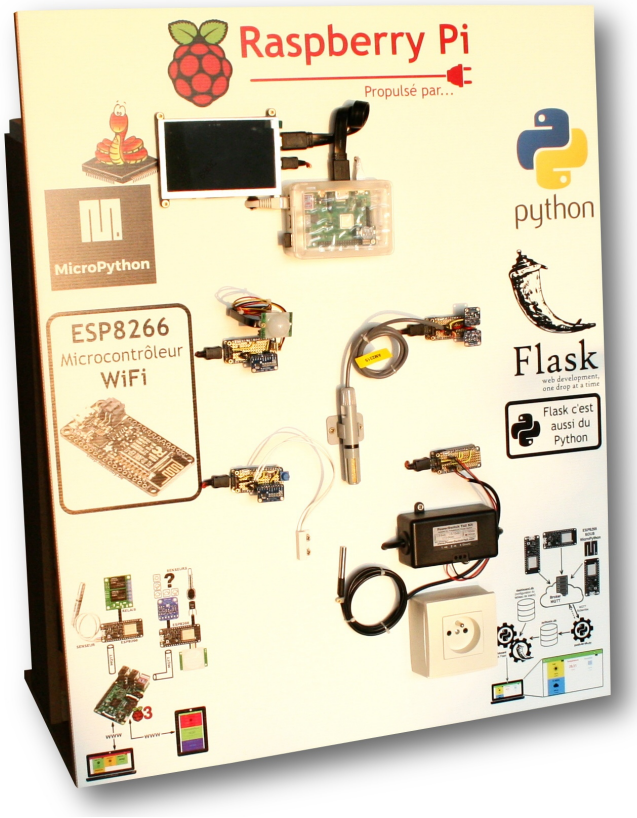
GREAT

```
rshell --port /dev/ttyUSB0 --baud 115200 --buffer-size 128 --editor nano
```

# Support matériel




## Support matériel sous MicroPython

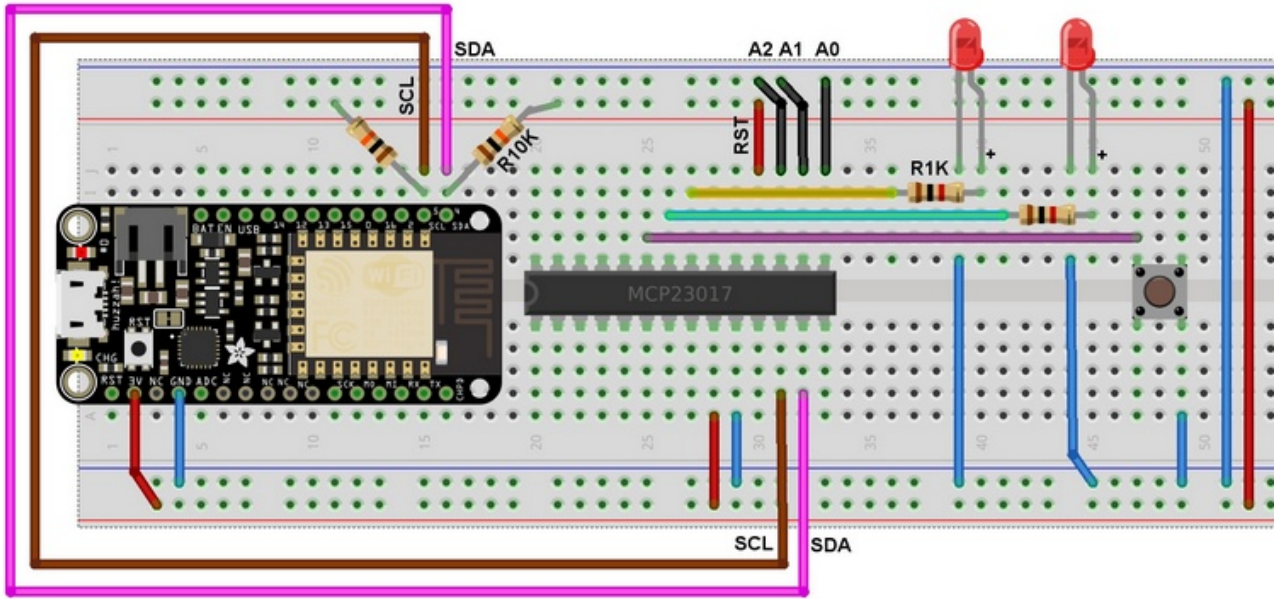
- Senseur les plus courants (BMP, BME, DHT, TSL, DS18B20, etc)
- Support d'afficheurs ( OLED, LCD, TFT, NeoPixel, etc )
- Adafruit Industries pour le support de ses produits sous CircuitPython → faire un retro-portage vers MicroPython
- La communauté MicroPython active

## Des pilotes pour MicroPython

- **GitHub : La-Maison-Pythonic**
  https://github.com/mchobby/la-maison-pythonic
- **GitHub : esp8266-upy**
  https://github.com/mchobby/esp8266-upy

# Dépasser les limites matérielles

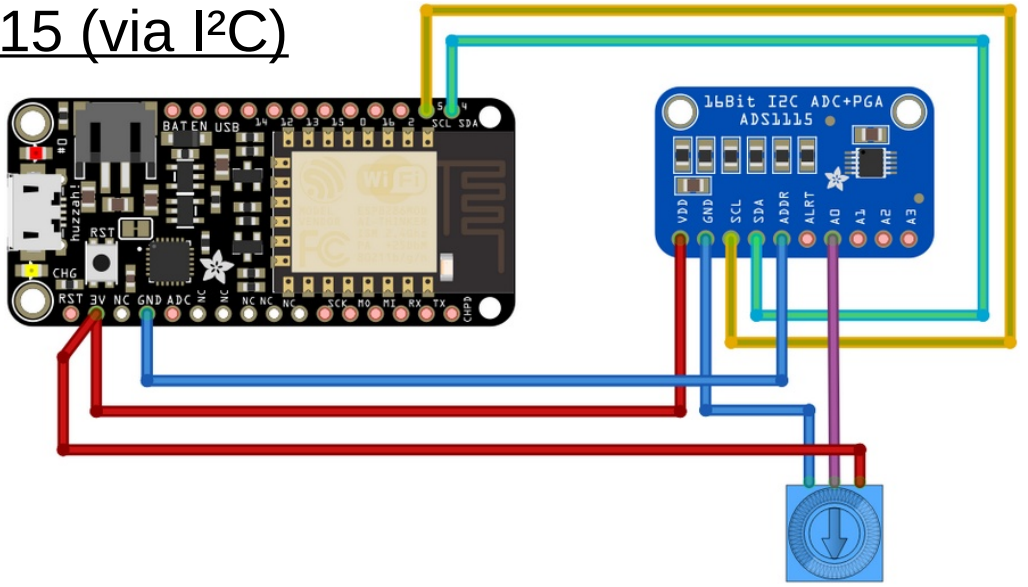
## Pas assez de GPIO ? → MCP23017 (via I²C)




```
from time import sleep
from machine import I2C, Pin
from mcp230xx import MCP23017

i2c = I2C( sda=Pin(4), scl=Pin(5), freq=20000 )
mcp = MCP23017( i2c=i2c )

mcp.setup( 0, Pin.OUT )
mcp.setup( 1, Pin.OUT )

for i in range( 10 ):
    # Activer sortie 0
    mcp.output( 0, True )
    sleep( 0.5 )
    mcp.output( 1, True )
    sleep( 0.5 )
    # Désactiver sortie 1
    mcp.output( 0, False )
    sleep( 0.5 )
    mcp.output( 1, False )
    sleep( 0.5 )
```

## Des entrées analogique ? → ADS1115 (via I²C)

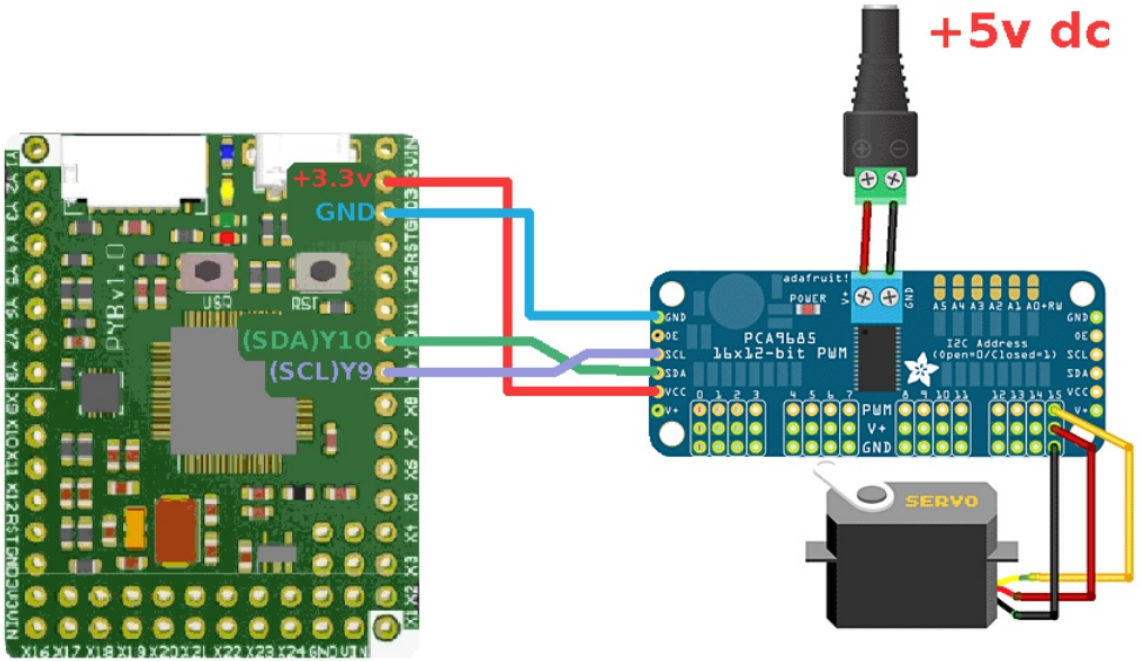
```
from machine import I2C, Pin
from ads1x15 import *
i2c = I2C( sda = Pin(4), scl=Pin(5) )
adc = ADS1115(i2c = i2c, address = 72, gain = 0)

value = adc.read( rate=0, channel1=0 )
print( value ) #  17549

print( value * 0.1875 / 1000 ) # 3.29025 V
```



# Dépasser les limites matérielles

## Des sorties PWM ? → pca9685 (via I²C)

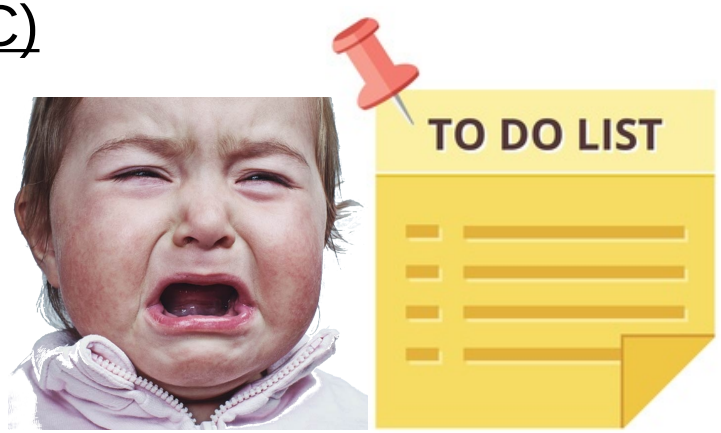
https://wiki.mchobby.be/index.php?title=MicroPython-PWM-DRIVER




```
from pyb import I2C
from servoctrl import ServoCtrl

# Initialise le bus I2C
i2c = I2C( 2, I2C.MASTER )

# Crée le controleur PWM.
driver = ServoCtrl( i2c )

# Angle servo moteur #15 = 45 degrés
driver.position( 15, 45 )

# Angle servo moteur #15 = 180 degrés
driver.position( 15, 180 )
```

## Des sorties analogiques ? → MCP4725 (via I²C)


Convertisseur Digital/Analogique pour Microcontrôleur, interface I2C, 12 bits



# ESP8266 sous MicroPython
# Difficultés rencontrées

**1) Ne pas oublier de réduire la taille de la mémoire tampon d'échange en communication série.**

La valeur par défaut est trop grande pour l'ESP8266 sous MicroPython.

```
rshell --port /dev/ttyUSB0 --baud 115200 --buffer-size 128 --editor nano
```

**2) Ne supporte pas encore le protocole mDns.**

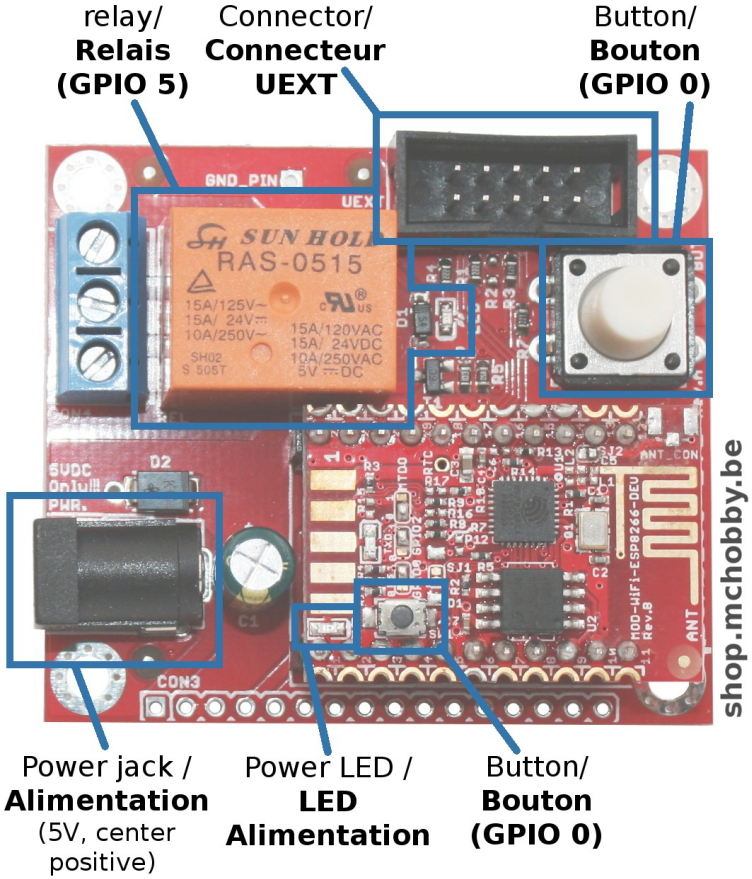
Celui-ci permet de résoudre des noms de machine comme « pythonic.local » sur les réseaux domestiques.
Les box internet offrent parfois ce service mais la stabilité est assez aléatoire dans le temps.

**Conséquence :** il faut utiliser une IP Fixe pour un « serveur » si celui-ci doit être contacté par l'ESP8266. Cfr : Livre « Python, Raspberry Pi et Flask ».

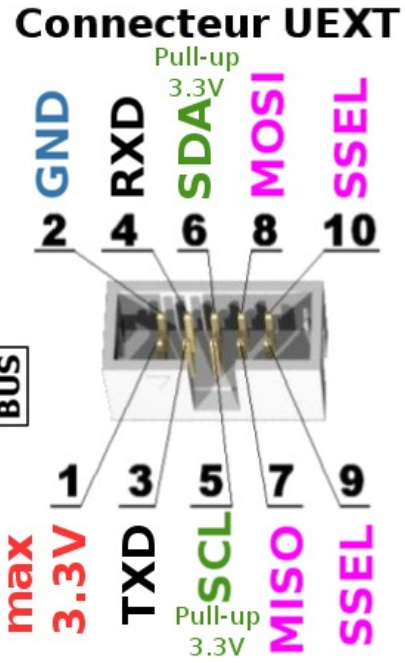
# Prototypage rapide et éducation

## L'ESP8266-EVB et **port UEXT** par OLIMEX

```
>>> from machine import Pin
>>> relay = Pin(5, Pin.OUT)
>>> relay.value( 1 ) # Relais activé
>>> relay.value( 0 ) # Relais inactif
>>> relay.value( 1 ) # Relais activé
>>> relay.value( 0 ) # Relais inactif
>>> relay.value( 1 ) # Relais activé
>>> relay.value( 0 ) # Relais inactif
```



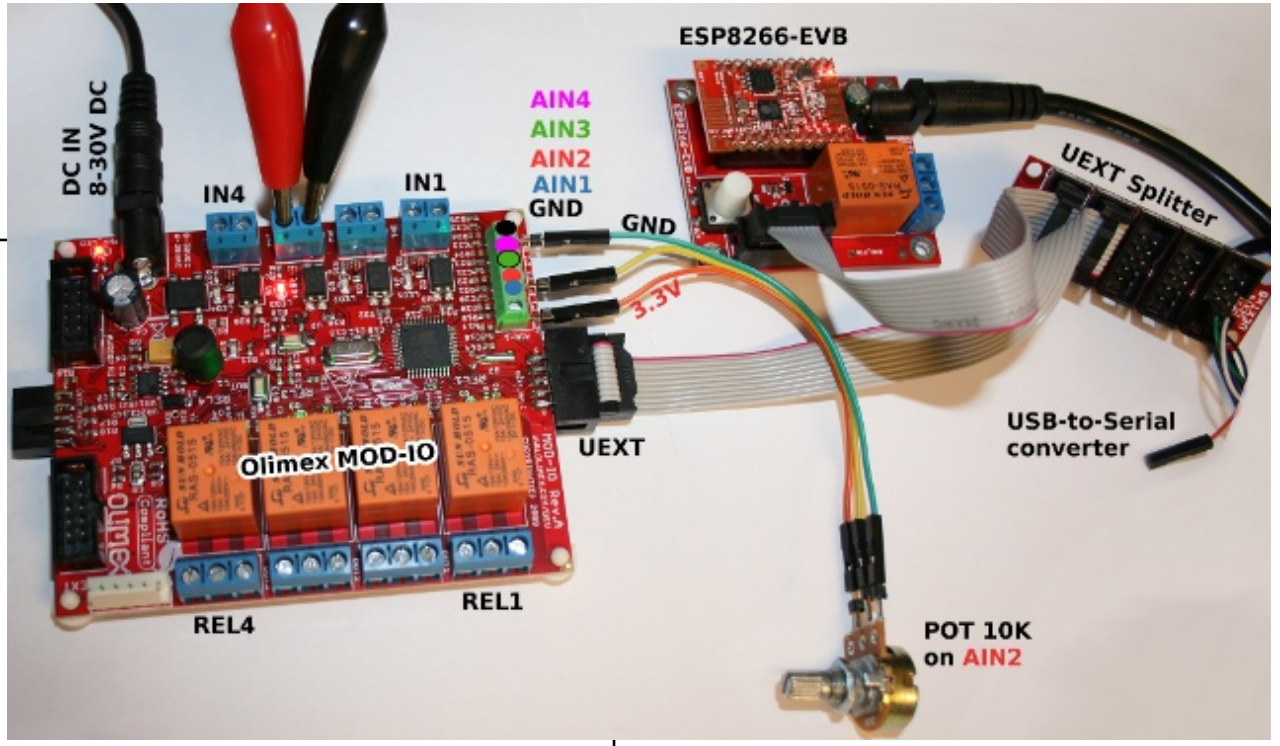

| ESP GPIO | Pin # | Signal Name | |
|---|---|---|---|
| | 1 | 3.3V max!! | |
| | 2 | GND | |
| GPIO 1 | 3 | TXD | |
| GPIO 3 | 4 | RXD | |
| GPIO 4 | 5 | SCL | I²C BUS |
| GPIO 2 | 6 | SDA | I²C BUS |
| GPIO 12 | 7 | MISO | SPI BUS |
| GPIO 13 | 8 | MOSI | SPI BUS |
| GPIO 14 | 9 | SCK (SCLK) | SPI BUS |
| GPIO 15 | 10 | SSEL (SS) | SPI BUS |




```
>>> from machine import Pin
>>> from time import sleep_ms
>>> btn = Pin( 0, Pin.IN, pull=Pin.PULL_UP )
>>> while True:
>>>     print( '--' if btn.value()==1 else 'PRESSE' )
>>>     sleep_ms( 300 )
```


ESP8266-EVB (carte d'évaluation) :
https://shop.mchobby.be/product.php?id_product=668
ESP8266-DEV (carte de développement) :
https://shop.mchobby.be/product.php?id_product=667
Gamme d'extension UEXT :
https://shop.mchobby.be/category.php?id_category=138

# ESP8266-EVB et pour UEXT




```
# Utilisation du MOD-IO d'Olimex
# avec un ESP8266 sous MicroPython
#
# GitHub: esp8266-upy//modio

from machine import I2C, Pin
from time import sleep_ms
from modio import MODIO

# Bus I2C sur le connecteur UEXT
i2c = I2C( sda=Pin(2), scl=Pin(4) )
brd = MODIO( i2c )

# Lire les entrées analogiques ===========
for input_index in range( 4 ):
    print( 'Analogique %s : %s Volts' % /
  (input_index,brd.analogs[input_index] ))

print( 'Lire toutes les entrées analogiques, mode RAW, en une seule opération' )
print( brd.analogs.states )

# === Lire les entrées Opto-Isolées ========================
print( brd.inputs.states )    # Lire toutes les entrées Opto-Isolées
print( brd.inputs[2] )        # Lire la 3ieme entrée Opto-Isolée

# === RELAIS ===================================
brd.relais[0] = True
brd.relais[2] = True
print( brd.relais.states ) # Etat des Relais[0..3]
brd.relais.states = False  # Eteindre tous les relais

print( "That's the end folks")
```

# Remplacer Arduino par MicroPython ?

- **Pour apprentissage : OUI**
  Le langage et le typage faible de MicroPython diminue le temps nécessaire avant les premières expériences.
- **Plateforme matérielle : OUI**
  MicroPython est parfait une fois les limites matérielles de l'ESP8266 dépassée en ajoutant des GPIOs et des entrées analogiques à l'aide de composants supplémentaires. Le support I2C, OneWire, SPI en fait un candidat de choix.
- **Point de vue librairie : OUI mais...**
  Les pilotes de composants sont nettement plus facile à trouver pour Arduino.
- **La communauté : OUI mais…**
  La communauté, bien qu'active, s'exprime principalement en anglais. Elle n'atteint cependant pas la taille de la communauté Arduino bien plus ancienne.
- **Stabilité : OUI**
  MicroPython existe depuis plusieurs années et démontre une stabilité exemplaire. Son cycle de développement se poursuit pour améliorer l'existant.
- **Puissance brute : a tester...**
  D'un côté nous avons un interpréteur bytecode sur un coeur 80 Mhz, de l'autre du C compilé sur un coeur 16 Mhz. Ce point reste à tester.
- **Faut-il remplacer Arduino ?**
  Pas forcement, apprendre la rigueur du C est important, voire essentiel.
  Par contre, être confronté à cette rigueur dans un second temps est peut être plus approprié…

*« guider avec bienveillance plutôt que subir les lois de façon arbitraire ! »*