

MicroPython  
Que faire avec un ESP32-C3 mini ?  
Anumby nov-2024

## Python

Python 2 → Python 3

Langage interprété (Basic, Javascript, ...)

Interpréteur CPython → shell Python, prompt >>>

Programme Python = « Script » Python

## Micropython

Interpréteur CPython « allégé » pour microcontrôleur (SOC)

Toutes les fonctionnalités de Python 3

Quelques librairies standards

Firmwares sur le site Micropython : <https://micropython.org/download/>

## Import simple

```
>>> import math
>>> math.sin(math.pi/2)
1.0
```

## Import avec alias

```
>>> import math as mt
>>> mt.sin(mt.pi/2)
1.0
```

## Import d'éléments isolés

```
>>> from math import pi, sin, cos, exp
>>> cos(pi/2)
6.123233995736766e-17
```

# Python - Librairie/module

## Afficher le contenu d'une librairie

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'e',
 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e',
 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'f',
 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',
 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf',
 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder',
 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

## Librairie time

```
>>> import time
>>> time.sleep(5)    # temps en s
```

## Ecriture d'une fonction

```
def moyenne(arg1, arg2):  
    m = (arg1 + arg2)/2  
    return m
```

## Appel de la fonction

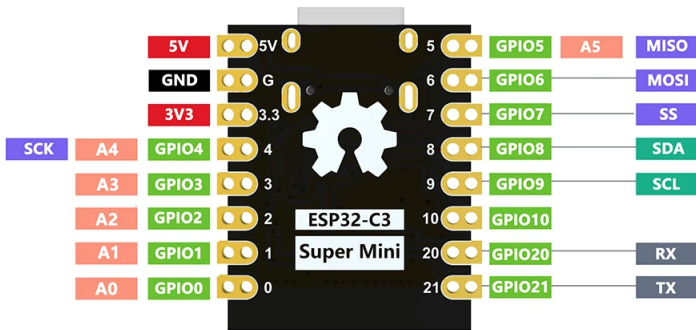
```
>>> moyenne(1.5, 7)  
4.25
```

## Important



Respecter l'indentation !

# Micropython - ESP32-C3 mini



■ Pin No.

■ Power

■ ADC

■ SPI

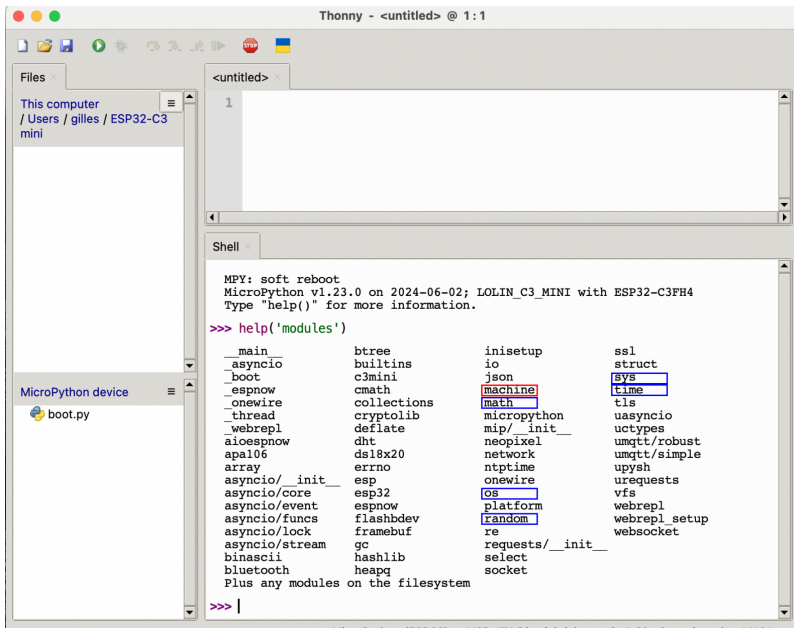
■ GND

■ UART

■ Digital

⚠ Circuit vu de dessous !

# MicroPython - ESP32-C3 mini



The screenshot shows the Thonny IDE interface. The top bar indicates the file is named "<untitled>" and is opened at a 1:1 scale. The left sidebar has two panes: "Files" and "MicroPython device". The "Files" pane shows the directory structure: "This computer" > "Users / gilles" > "ESP32-C3 mini". The "MicroPython device" pane shows a file named "boot.py". The main editor area is titled "<untitled>" and contains a single line of code: "1". The bottom pane is the "Shell", which displays the following text:

```
MPY: soft reboot
MicroPython v1.23.0 on 2024-06-02; LOLIN_C3_MINI with ESP32-C3FH4
Type "help()" for more information.

>>> help('modules')

   _main_          btree          inisetup          ssl
   _asyncio        builtins        io                struct
   _boot           c3mini          json              sys
   _espnw          cmath           machine          time
   _onewire        collections     math             tls
   _thread         cryptolib      micropython       uasyncio
   _webrepl        deflate        mip/_init_        uctypes
   aioespnw        dht            neopixel          umqtt/robust
   apal06          ds18x20        network           umqtt/simple
   array           errno          ntp               upysh
   asyncio/_init_  esp            onewire           urequests
   asyncio/core    esp32          os               vfs
   asyncio/event   espnw          platform          webrepl
   asyncio/functs  flashbdev     random           webrepl_setup
   asyncio/lock    framebuf      re                websocket
   asyncio/stream  gc            requests/_init_
   binascii        hashlib       select
   bluetooth       heapq         socket
   Plus any modules on the filesystem

>>> |
```

## Librairie machine

Fonctions liées à la gestion du hardware, en particulier les GPIOs

```
>>> import machine
>>> dir(machine)
['__class__', '__name__', 'ADC', 'ADCBlock', ...]
```

## GPIO

- module Pin : contrôle des pins en entrée/sortie
- module PWM : Pulse Width Modulation
- module Timer : contrôle des timers hardware
- module UART : bus de liaison série
- module I2C : bus de liaison I2C
- module ADC : conversion analogue → digital
- ...

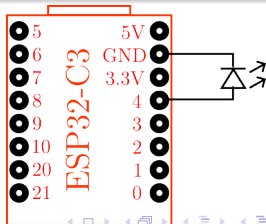


## Module Pin : output

```
>>> from machine import Pin
>>> p4 = Pin(4, Pin.OUT)    # pin 4 utilisée en sortie
>>> dir(p4)
['___class__', 'value', ...
 ..., 'board', 'init', 'irq', 'off', 'on']
>>> p4.value(0)            # p4 à 0V,  identique à p4.off()
>>> p4.value(1)            # p4 à 3.3V, identique à p4.on()
```

### Exercise

Ecrire un script qui fait clignoter la LED 10 fois avec une période de 1s et un rapport cyclique de 0.5.  
Idem avec la LED 'onboard' (GPIO 8)

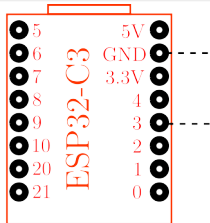


## Module Pin : input

```
>>> from machine import Pin
>>> p3 = Pin(3, Pin.IN)    # pin 4 utilisée en entrée
>>> p3.init(pull = Pin.PULL_DOWN)    # 0 si pas d'input
ou
>>> p3.init(pull = Pin.PULL_UP)      # 3.3V si pas d'input
puis
>>> p3.value()    # lecture de la valeur
1
```


## Exercice

Configurer p3 en mode PULL\_UP et afficher sa valeur dans la console toutes les secondes.  
Idem avec le bouton BOOT de la carte (GPIO 9).



## Module Pin : interruption

Principe : le changement d'état de la pin déclenche l'appel (quasi)instantanée d'une fonction ('handler')

transition 0 → 1  Pin.IRQ\_RISING

transition 1 → 0  Pin.IRQ\_FALLING

### Exemple

```
def mon_handler(p):  
    print('pin 3')  
  
>>> p3 = Pin(3, Pin.IN, Pin.PULL_DOWN)  
>>> p3.irq(trigger=Pin.IRQ_RISING, handler=mon_handler)  
<IRQ>
```

## Librairie time

<code>sleep(2)</code>	pause de l'exécution pendant 2 s
<code>sleep_ms(10)</code>	pause de l'exécution pendant 10 ms
<code>sleep_us(100)</code>	pause de l'exécution pendant 100 $\mu$ s
<code>time()</code>	valeur du compteur des secondes (origine arbitraire)
<code>ticks_ms()</code>	valeur du compteur des ms
<code>ticks_us()</code>	valeur du compteur des $\mu$ s

## Exercice

En utilisant la fonction `ticks_ms` de la librairie `time`, modifier la fonction `my_handler` pour supprimer les rebonds à la fermeture de l'interrupteur (debouncing).

## Librairies os et sys

- `os` : commandes Linux du système de fichiers  
`remove`, `chdir`, `getcwd`, `listdir`, `mount`, `rmdir`, ...
- `sys` : fonctions système spécifiques  
`sys.path` : liste des chemins d'accès aux librairies (commande `import`)  
à mettre à jour lorsqu'on ajoute des répertoires avec des nouvelles librairies

## Scripts de démarrage

2 scripts lancés successivement au boot :

- `boot.py` : script de configuration  
contient toutes les options de configuration de l'utilisateur (`imports`, définition de fonctions, de variables, chemin vers les librairies, ...)
- `main.py` : script de lancement de la tâche principale  
boucle de lecture des capteurs, commande des actionneurs, ...

## Lancement d'un script au démarrage

Plusieurs options :

- copier le code du script dans `main.py`
- dans `main.py`, ajouter la ligne :

```
import mon_script
```

où `mon_script.py` est le nom du fichier script. S'il n'est pas dans le répertoire racine, son chemin d'accès doit être dans la liste `sys.path`

- dans `main.py`, ajouter la ligne :

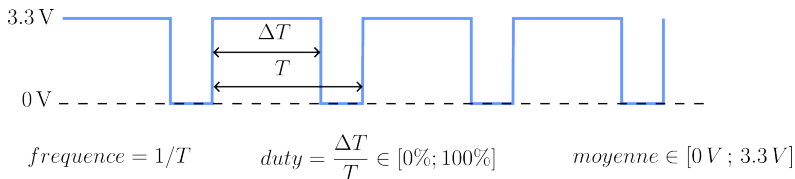
```
execfile(filename)
```

où `filename` est une chaîne de caractères contenant le nom complet du fichier (avec le répertoire et l'extension `.py`)

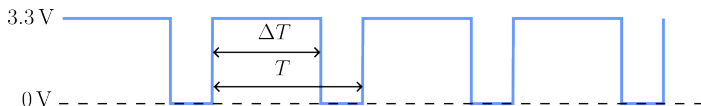
### Exercice

Ecrire un script qui fait clignoter la LED (GPIO 8) et le lancer au boot.

## Module PWM : Pulse Width Modulation



## Module PWM : Pulse Width Modulation



$$frequency = 1/T$$

$$duty = \frac{\Delta T}{T} \in [0\%; 100\%]$$

$$moyenne \in [0\text{ V} ; 3.3\text{ V}]$$

### Applications :

- commande de moteur à cc
- commande de moteur pas à pas
- commande de moteur brushless
- commande de servomoteur
- commande intensité LED

...



## Exemple

```
>>> from machine import PWM
>>> p4 = PWM(4, freq=2000, duty=800) # duty: 0 -> 1023 (100\%)
>>> p4.freq(1000) # changer la fréquence
>>> p4.duty(350) # changer le rapport cyclique
>>> p4.freq() # renvoie la fréquence
1000
>>> p4.duty() # renvoie le rapport cyclique
350
```

On peut augmenter la résolution sur le rapport cyclique

```
>>> p4.duty_ns(50000) # rapport cyclique en ns (50000ns = 50us)
>>> p4.duty_u16(15000) # duty : 0 -> 65535 = 2**16-1(100\%)
```

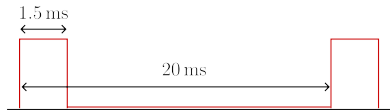
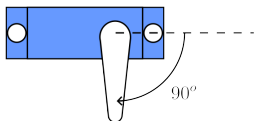
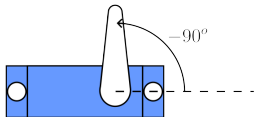
⚠ Pour l'ESP32, fréquence  $\geq 5$  Hz (période  $\leq 0,2$  s)

## Exercice

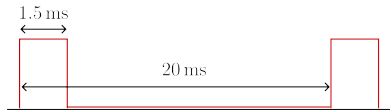
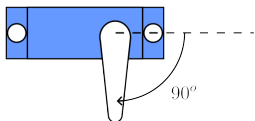
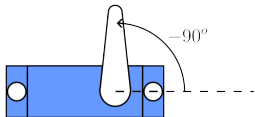
Une LED alimentée par une tension PWM clignote à la fréquence du signal. Pour les fréquences supérieures à quelques dizaines de hertz, l'oeil ne perçoit pas le clignotement et voit un éclaircissement moyen.

Ecrire un script qui fait passer progressivement l'éclaircissement de la LED de 0 à 100% en 2s, puis éteint la LED.

## Application : commande de servomoteur SG90



## Application : commande de servomoteur SG90



## Exercice

Ecrire un script qui envoie un signal PWM sur la commande (cmd), et qui définit une fonction `angle(x)` qui positionne le servo à la position  $x \in [-90^\circ; 90^\circ]$

## Module Timer

Un timer provoque l'appel, périodiquement ou une seule fois, d'une fonction définie par l'utilisateur (ISR ou callback). Si un script est en cours d'exécution, il est interrompu et il reprend à la fin de l'appel.

### Applications

- surveillance d'un capteur
- rafraîchissement d'un affichage
- commande d'un moteur pas à pas
- ...

## Exemple

```
def compteur(t):  
    global n  
    print(n)  
    n += 1  
  
>>> from machine import Timer  
>>> tim = Timer(0, period=1000, callback=compteur)  
>>>  
0  
1  
...  
>>> tim.deinit()    # pour désactiver le timer
```

## Attention

- la fonction callback prend un argument (timer t)
- pour l'ESP32C3 mini, les numéros de Timer sont toujours pairs : 0, 2, 4, ...

## Exercice

Ecrire une fonction `blink` qui change l'état de la LED onboard (GPIO 8), puis créer un timer qui utilise cette fonction comme callback pour faire clignoter la LED.

## Système de fichiers

### Deux types de fichiers

- fichier texte : contient des chaînes de caractères encodées en utf8. Les lignes se terminent par '`\r\n`' (retour chariot et nouvelle ligne).
- fichier binaire : peut contenir n'importe quelle suite d'octets.

### Création d'un fichier texte

```
>>> fd = open('test.txt', 'w') # 'w'= fichier texte ouvert
                                # en écriture
>>> fd.write('debut\r\n')      # 1ère ligne
8
>>> fd.write('ceci est un test\r\n') # 2e ligne
18
>>> fd.write('fin\r\n')        # 3e ligne
5
>>> fd.close()                # fermeture du fichier
```



## Lecture d'un fichier texte existant

```
>>> fd = open('test.txt', 'r') # 'r'= fichier texte ouvert
                                # en lecture
>>> fd.readline()             # on lit la 1ère ligne
'debut\r\n'
>>> fd.readline()             # on lit la 2e ligne
'ceci est un test\r\n'
>>> fd.readline()             # on lit la 3e ligne
'fin\r\n'
>>> fd.close()                 # fermeture du fichier
```

ou bien

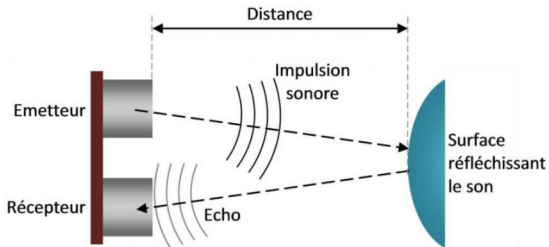
```
>>> fd.readlines().           # on lit toutes les lignes d'un coup
['début\r\n', 'ceci est un test\r\n', 'fin\r\n']
>>> fd.close()
```

## Exercice

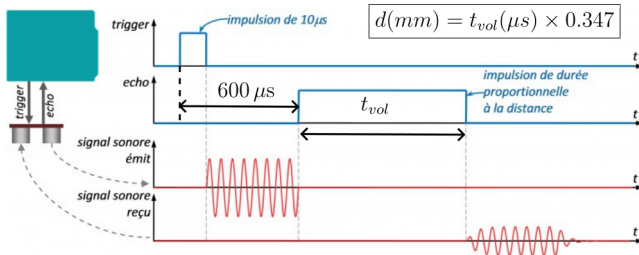
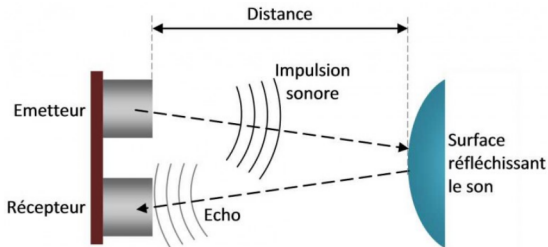
Ecrire un script qui enregistre dans un fichier texte, chaque seconde et pendant 10s, la date, l'heure et la température du microcontrôleur (une ligne par enregistrement).

Pour l'horodatage, utiliser la fonction `localtime` de la librairie `time`. La température est donnée par la fonction `mcu_temperature` (sans accent !) de la librairie `esp32`.

## Capteur de distance ultrasons HC-SR04



# Capteur de distance ultrasons HC-SR04



## Fonction mesure de distance

```
from machine import Pin
from time import sleep_us, ticks_us

# initialisation pins et constantes
trig = Pin(...)
echo = Pin(...)
cson = 0.347      # vitesse du son en mm/us

# fonction de lecture du capteur
def mesure()
    génération de l'impulsion trigger
    attente front montant echo
    déclenchement chrono
    attente front descendant echo
    arrêt chrono
    conversion temps -> distance
    renvoyer le résultat
```