

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

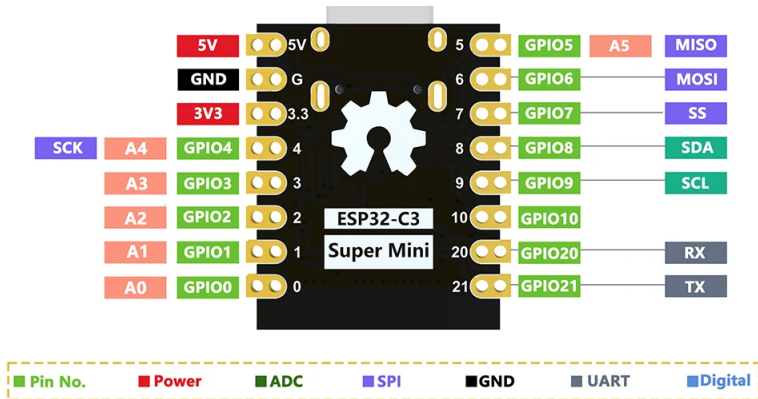
```
>>> moyenne(1.5, 7)  
4.25
```

Important



Respecter l'indentation !

Micropython - ESP32-C3 mini



⚠ Circuit vu de dessous !

MicroPython - ESP32-C3 mini

Thonny - <untitled> @ 1:1

Files

This computer
/ Users / gilles / ESP32-C3 mini

<untitled>

1

Shell

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

```
>>> help('modules')
```

<code>__main__</code>	<code>btree</code>	<code>inisetup</code>	<code>ssl</code>
<code>__asyncio</code>	<code>builtins</code>	<code>io</code>	<code>struct</code>
<code>__boot</code>	<code>c3mini</code>	<code>json</code>	<code>sys</code>
<code>__espnow</code>	<code>cmath</code>	<code>machine</code>	<code>time</code>
<code>__onewire</code>	<code>collections</code>	<code>math</code>	<code>tls</code>
<code>__thread</code>	<code>cryptolib</code>	<code>micropython</code>	<code>uasyncio</code>
<code>__webrepl</code>	<code>deflate</code>	<code>mip/_init__</code>	<code>uctypes</code>
<code>aiospnow</code>	<code>dht</code>	<code>neopixel</code>	<code>umqtt/robust</code>
<code>apal06</code>	<code>ds18x20</code>	<code>network</code>	<code>umqtt/simple</code>
<code>array</code>	<code>errno</code>	<code>ntptime</code>	<code>upysh</code>
<code>asyncio/_init__</code>	<code>esp</code>	<code>onewire</code>	<code>urequests</code>
<code>asyncio/core</code>	<code>esp32</code>	<code>os</code>	<code>vfs</code>
<code>asyncio/event</code>	<code>espnow</code>	<code>platform</code>	<code>webrepl</code>
<code>asyncio/funcs</code>	<code>flashbdev</code>	<code>random</code>	<code>webrepl_setup</code>
<code>asyncio/lock</code>	<code>framebuf</code>	<code>re</code>	<code>websocket</code>
<code>asyncio/stream</code>	<code>gc</code>	<code>requests/_init__</code>	
<code>binascii</code>	<code>hashlib</code>	<code>select</code>	
<code>bluetooth</code>	<code>heapq</code>	<code>socket</code>	

Plus any modules on the filesystem

```
>>> |
```

MicroPython device

boot.py

Librairie machine

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

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

GPIO

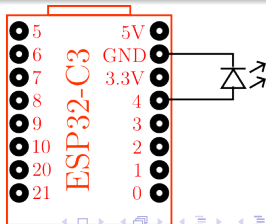
- module Pin : contrôle des pins en entrée/sortie
- module PWM : Pulse Width Modulation
- module Timer : contrôle des timers hardware
- module UART : bus de liaison série
- module I2C : bus de liaison I2C
- module ADC : conversion analogue → digital
- ...

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()
```

Exercice

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)

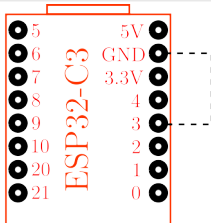


Module Pin : input

```
>>> 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 μ 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 μ 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 :

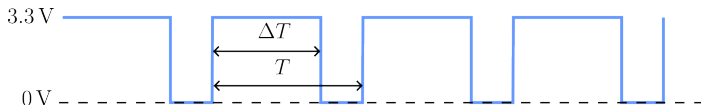
```
execfile(filename)
```

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

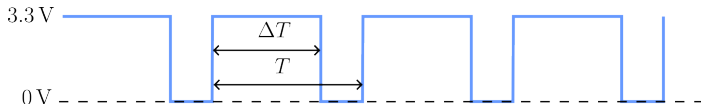


$$frequency = 1/T$$

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

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

Module PWM : Pulse Width Modulation



$$frequency = 1/T$$

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

$$moyenne \in [0V ; 3.3V]$$

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\%)
```



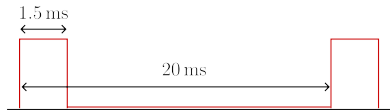
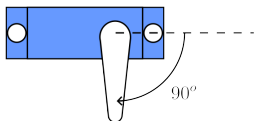
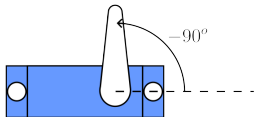
Pour l'ESP32, fréquence ≥ 5 Hz (période $\leq 0,2$ s)

Exercice

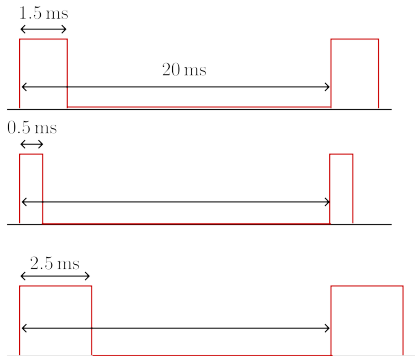
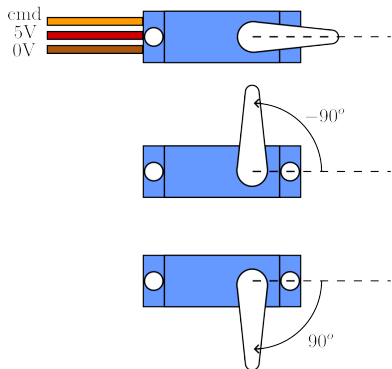
Une LED alimentée par une tension PWM clignote à la fréquence du signal. Pour les fréquences supérieures à quelques dizaines de hertz, l'œil ne perçoit pas le clignotement et voit un éclaircissement moyen.

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

Écrire 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  
>>> n = 0  
>>> tim = Timer(0, period=1000, callback=compteur)  
>>>  
0  
1  
...  
>>> tim.deinit()    # pour désactiver le timer
```

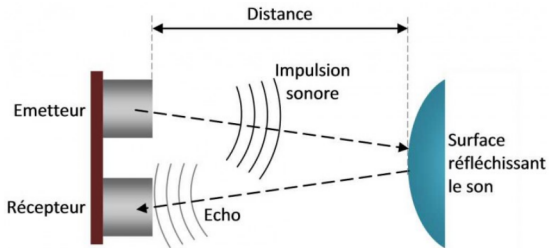
Attention

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

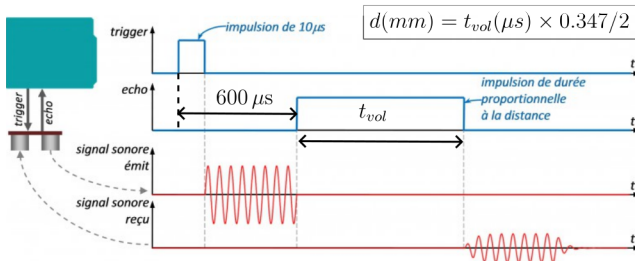
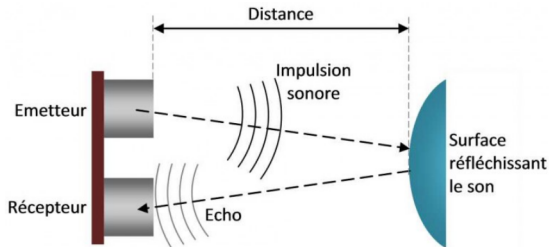
Exercice

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.

Capteur de distance ultrasons HC-SR04



Capteur de distance ultrasons HC-SR04



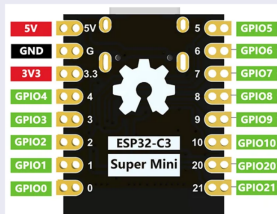
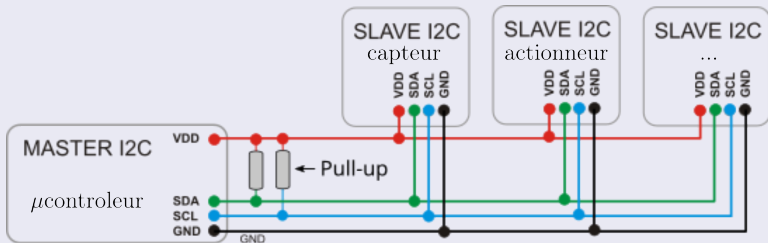
Fonction mesure de distance

```
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
```

Module I2C (Inter Integrated Circuit)



adressage 7 bits \rightarrow 128 périphériques

débit \sim 400 kB/s

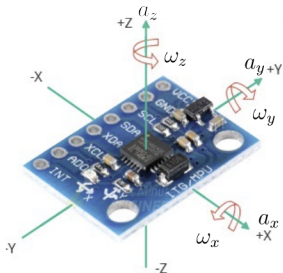
sda et scl : 2 pins quelconques

Applications : capteur de température/pression, accéléromètre, centrale inertielle (IMU), driver moteur/servomoteurs, afficheur OLED, ...

Remarque : adresse I2C de l'esclave imposée par le fabricant → impossible de mettre 2 périphériques identiques sur le même bus.

```
>>> from machine import SoftI2C
>>> i2c = SoftI2C(sda=4, scl=3)
>>> i2c.scan()
[104]          (liste des adresses des périphériques trouvés
                sur le bus)
```

MPU6050 - Présentation



Algorithme de fusion

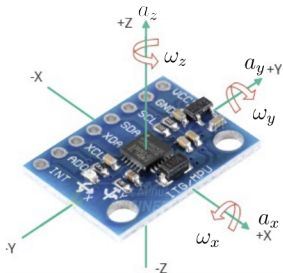
accélération linéaire \vec{a}

champ de gravité \vec{g}

position angulaire :

- roulis, tangage, lacet
- angles d'Euler

MPU6050 - Présentation



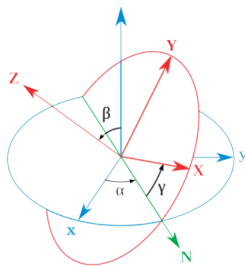
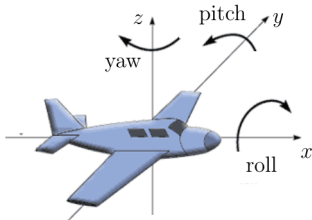
Algorithme de fusion

accélération linéaire \vec{a}

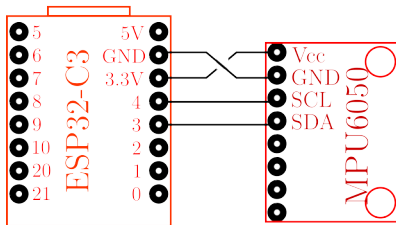
champ de gravité \vec{g}

position angulaire :

- roulis, tangage, lacet
- angles d'Euler



MPU6050 - Librairie MPU6050dmp20



```
>>> from machine import SoftI2C
>>> from MPU6050dmp20 import *
>>> i2c = SoftI2C(sda=3, scl=4)
>>> mpu = MPU6050dmp(i2c)
***** MPU6050dmp init : MPU6050 found address 0x68
>>>
```

Mesure des offsets

```
>>> mpu.calibrate()
***** MPU6050 calibration init - Hit CTRL-C to stop
ax0ff=-4523, ay0ff=507, az0ff=620, gx0ff=130, gy0ff=24, gz0ff=39
...
***** MPU6050 calibration interruption
Offsets set to ax0ff=-4452, ay0ff=511, az0ff=628, gx0ff=130, gy
>>>
```

Rebooter l'ESP32 (Ctrl-D)

```
>>> from machine import SoftI2C
>>> from MPU6050dmp20 import *
>>> i2c = SoftI2C(sda=3, scl=4)
>>> mpu = MPU6050dmp(i2c, ax0ff=-4452, ay0ff=511, az0ff=628,
                      gx0ff=130, gy0ff=26, gz0ff=39)
***** MPU6050dmp init : MPU6050 found address 0x68
>>> mpu.dmpInitialize()
>>> mpu.setDMPEnabled(True)
>>> mpu.getIntStatus()
```


Après initialisation du dmp, données mises à jour toutes les 10 ms (100 Hz)

```
def getData():
    mpu.resetFIFO()
    mpu.getIntStatus()
    while mpu.getFIFOCount() != 42:
        if mpu.getFIFOCount() > 42:
            mpu.resetFIFO()
    buf = mpu.getFIFOBytes(42)
    quat = mpu.dmpGetQuaternion(buf) # quaternion
    acc = mpu.dmpGetFifoAccel(buf)    # accel - grav
    gyro = mpu.dmpGetFifoGyro(buf)    # vitesse angulaire
    grav = mpu.dmpGetGravity(quat)    # gravité
    linac = mpu.dmpGetLinearAcc(grav, acc) # accélération
    yaw, pitch, roll = mpu.dmpGetYawPitchRoll(quat, grav)
    theta, phi, psi = mpu.dmpGetEuler(quat)
    return yaw, pitch, roll
```

Système de fichiers

Deux types de fichiers

- fichier texte : contient des chaînes de caractères encodées en utf-8. 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`.

module UART

Protocole de liaison série full duplex

Chaines de caractères <class 'str'>

```
>>> s = 'a b_c-d'      # ou s = "a b_c-d"
>>> len(s)             # longueur
7
>>> s[0]               # 1er caractère
'a'
>>> s[1]               # 2e caractère
' ',
>>> s[-1]              # dernier caractère
'd'
>>> s[2:5].            # sous-chaine
'b_c'
>>> s[2] = 'x'
TypeError: 'str' object does not support item assignment
```

⚠ une chaîne de caractères est un objet *non mutable*

Chaines d'octets <class 'bytes'>

```
>>> r = b'\x00\x01\x02'
>>> len(r)          # longueur
3
>>> r = b'\x61\x0d\x0a'
b'a\r\n'
>>> r[0]
97          (0x61)
>>> r[1]
13          (0x0d)
>>> r[2]
10          (0x0a)
```

Une chaîne d'octets est une **liste d'octets** → chaque élément de la liste est **un entier compris entre 0 et 255**

```
>>> r[1] = 10
```

```
TypeError: 'bytes' object does not support item assignment
```

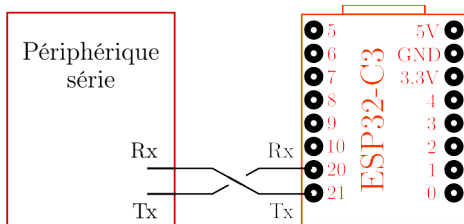
⚠ une chaîne de bytes est un objet *non mutable*

Encodage et décodage (utf-8)

chaîne de caractères	$\xrightarrow{\text{encodage}}$	chaîne d'octets
chaîne de caractères	$\xleftarrow{\text{decodage}}$	chaîne d'octets

```
>>> s = 'première ligne\r\n'
>>> s.encode()
b'premi\xc3\xa8re ligne\r\n'
>>> r = b'\x0d\x0a'
>>> r.decode()
'\r\n'
```

Module UART

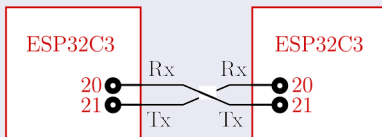


```
>>> from machine import UART
>>> u = UART(1, rx=20, tx=21)
>>> u
UART(1, baudrate=115211, bits=8, parity=None, stop=1, tx=21,
      rx=20, rts=-1, cts=-1, txbuf=256, rxbuf=256,
      timeout=0, timeout_char=0)
>>> u.write('chaine de caractères envoyée')
30          (nombre de caractères envoyés)
```



```
>>> u.read()    # lit le contenu du buffer d'entrée
b"chaîne d'octets re\xc3\xa7ue"
>>> u.read(10)  # lit au plus 10 octets dans le buffer d'entrée
>>> u.any()     # nombre d'octets en attente de lecture
```

Exercice : communication série entre deux ESP32C3



Etablir une connexion série entre les deux ESP32 et tester l'émission et la réception des deux côtés.

Allumer/éteindre la diode GPIO8 d'un ESP32 à partir de l'autre.

`exec(cmd)` → execute la commande `cmd` (chaîne de caractères ou d'octets)