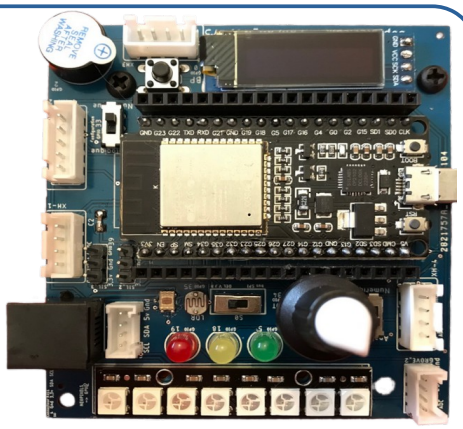
Document d'information à la date du 18/12/2025 / Contact : jacques.chouteau@ac-nantes.fr

Carte à microcontrôleur FabLab



Niveaux ciblés :
- Seconde SNT : Objets connectés, géolocalisation, réseau
- Spécialité NSI : Divers notions en référence au BO *(Cf page 2)*
- Physique - Chimie : Intégration d'outils numériques

Points clés :
- Embarquer un vrai interpréteur Python (micropPython),
- Réaliser des projets sans câblage, et possibilité de connecter des extensions *(capteurs, actionneurs, ...)*
- Aborder l'apprentissage du langage Python par la pratique.

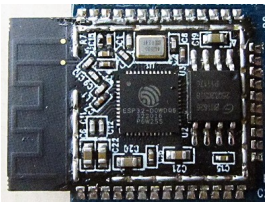
Cette carte FabLab – SNT – NSI – est la 3eme version après 3 ans d'utilisation en seconde SNT, ainsi qu'en spécialité NSI *(niveau 1ere et terminale)* ainsi qu'en physique – chimie.

Les choix des périphériques embarqués ont été déterminés dans la perspective d'**aborder par la pratique** des notions d'informatique au programme de SNT et NSI *(algorithme, apprentissage du langage Python, …)* et faciliter la réalisation de projets **sans avoir à câbler des périphériques sur des plaques de prototypage ni acheter des extensions**.

Téléchargement des pilotes pour communiquer avec la carte

https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

Microcontrôleur :



ESP32 : Espressif – Version WROOM 32 D
32 bits – 240 MHz – Dual Core – 520 KB SRAM – 448 KB flash ROM
WiFi / Bluetooth
GPIO : Conversions Analog. → Numériq. et Numériq. → Analog. / Capacitive touch / PWM / ...
Bus d'extension pour développement de modules supplémentaires : I2C / SPI
...
Référence : https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf

Caractéristiques de la carte SoproLab – SNT – NSI -

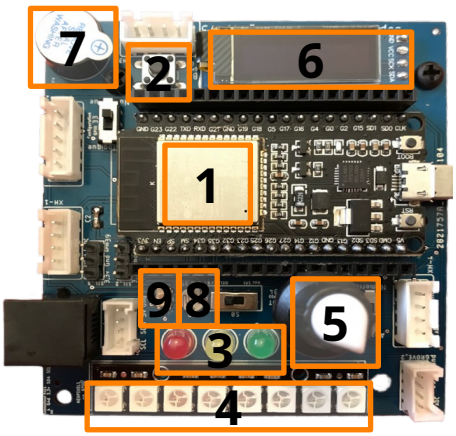
L'ESP32 permet d'**embarquer un vrai interpréteur Python (microPython)**. Certaines instructions sont cependant spécifiques aux microcontrôleurs (utilisation des GPIO, interruptions, PWM, ...) et certaines bibliothèques ne sont pas disponibles (matplotlib, …) mais la majorité des instructions ont été implémentées dans le langage micropython.

https://docs.micropython.org/en/latest/

L'ESP32 : un objet connecté par nature :

L'ESP32 dispose d'une interface WiFi et permet donc de développer un microserveur WEB.
L'interface WiFi permet soit de se connecter à un réseau local, soit de créer un point d'accès (avec la limite de 5 utilisateurs simultanés). L'adresse IP du microcontrôleur peut être affichée sur l'écran intégré à la carte et les élèves peuvent ensuite se connecter et interagir avec le microcontrôleur via un navigateur.

Document d'information à la date du 18/12/2025 / Contact : jacques.chouteau@ac-nantes.fr



1	ESP32 - WOOM - 32D
2	Bouton poussoir - *GPIO 0 - (1 : relâché / 0 : enfoncé)*
3	3 leds : verte - jaune - rouge *(GPIO 5 - 18 - 19)*
4	Ruban de 8 leds NeoPixel *(GPIO 12)*
5	Potentiomètre *(GPIO 34)*
6	Écran OLED 128x32 pixels *(bus I2C)*
7	Buzzer passif *(GPIO 2)*
8	LDR *(capteur de lumière analogique) (GPIO 35)*
9	TSL2561 *(capteur de lumière numérique) (bus I2C)*

Périphériques embarqués sur la carte :

La diversité des périphériques embarqués sur la carte permet de varier les scenarii pédagogiques ce qui facilite la pédagogique individualisée dans la mesure où il devient facile de proposer des approfondissements selon le degré d'avancement des élèves.

Périphériques	**Quelques références aux BO de SNT et ou NSI**
Ecran OLED 127x32 pixels *(interface I2C)*	Afficher des messages → **Interface Homme Machine** (IHM) Permet aussi d'afficher l'**adresse IP** du microcontrôleur (Objets connectés).
Bouton poussoir	Interaction avec l'utilisateur : lancer une séquence de mesure, … **(IHM)** Aborder les **boucles tant que** : *tant que le bouton poussoir n'est pas enfoncé …* Introduire la notion d'**interruption** matérielle *(principe développé dans le contexte de gestion des processus).*
3 LEDS Verte, Jaune et Rouge.	**Blocs d' instructions conditionnelles** : selon la valeur du signal produit par un capteur intégré à la carte (Cf LDR) allumer ou éteindre les leds. **Introduire la P.O.O. :** trois instances de classe Led( ).
Potentiomètre	**Principe de la conversion analogique → numérique**. Utilisation dans le cadre d'une IHM
Ruban de LED NeoPixel	La couleur d'une LED NeoPixel est déterminée selon le niveau R,V,B compris entre 0 et 255. On peut facilement ici faire le lien avec les pixels d'une image. En NSI, on met ici en application **les tuples** (R,V,B), **les listes** puisque chaque led est accessible selon son indice dans une liste, … et donc la mise en pratique **des boucles bornées** : *Pour i allant de 0 à 7 faire ...*
Buzzer passif	**Produire des sons de fréquence variable**. On peut soit l'utiliser comme instance de la classe PWM en déterminant la fréquence *(méthode freq() de la classe PWM)* mais c'est aussi le moyen d'utiliser les temporisations en microsecondes pour calculer les périodes et donc fréquences d'un son (programme sur le son en 2nde GT Physique-Chimie).
LDR Light Dépendent Résistor	Capteur de lumière résistif. Conversion analogique → numérique.
Capteur de lumière numérique TSL2561	Capteur de lumière numérique *(protocole I2C)*. *La documentation technique indique comment* ***calculer le niveau de luminosité en Lux à partir du niveau de luminosité mesuré***. On développe alors le principe de **mise en fonction** avec une valeur entrée en argument et une autre en sortie. Utilisation d'une bibliothèque spécifique au capteur (modularité du code) De plus cela permet de **comparer la différence entre un capteur numérique et un capteur analogique** (principe de l'étalonnage, sensibilité, linéarité, ...)

Document d'information à la date du 18/12/2025 / Contact : jacques.chouteau@ac-nantes.fr

Connecteurs pour extensions selon les projets :

1 connecteur 5p en XH2.5	5V / GND / GPIO 14 et 15 booléens / GPIO 33 Conv. Analog→Digit ou Bool	
1 connecteur 4p XH2.5 recopié en vers. GROVE	5V / GND / GPIO 25 Booléen / GPIO 32 Conv. Analog→Digit. ou Bool.	
1 connecteur 4p XH2.5	5V / GND / GPIO 13 et 27	
1 connecteur 4p XH2.5	5V / GND / GPIO 16 et 17 → connexion UART 2 (liaison série)	
1 connecteur GROVE	5V / GND / Bus I2C (SDA / SCL)	
2 connecteurs SIL 3p	3.3V / GND / GPIO 39	3.3V / GND / GPIO 26
1 connecteur RJ11 - 6p	3.3V / GND / Bus I2C / GPIO 4 et 36 (Conv. Analog. Numériq.)	
2 connecteurs SIL 19 de chaque côté de l'ESP32	Permet d'accéder à l'intégralité des broches du module ESP32 VROOM 32 D pour y implanter une carte d'extension.	

1 Connecteur USB-C : Connecteur de programmation, d'alimentation et communication avec l'ordinateur.

Quelques extensions déjà réalisées pour mettre les élèves en projet :

Développer une interface WEB et visualiser en temps réel l'évolution de la mesure d'une grandeur :
- → *html / css / javascript*
- → *Communication réseau via un socket et requettes Ajax pour la mise à jour*

Récepteur GPS
- → Étude de trame NMEA
- → Géolocalisation et positionnement sur une carte (librairie Folium)

Jeu de Simon avec 3 boutons poussoir et 3 leds

Séquence aléatoire visuelle (Led) et sonore à reproduire avec les boutons poussoir
- → paramétrage des GPIO en entrée ou sortie
- → programmation soit avec des dictionnaires (1ere NSI) ou en POO (Terminale)

Potentiomètre à glissière et afficheur 4 digit
- → Conversion analogique numérique
- → Exploitation d'une bibliothèque pour afficher une valeur sur un afficheur 4 Digit
- → Organiser une séquence de mesure (déclenchement, intervalles, transfert des données, ...)

Sonde de température DS18B20 (capteur numérique)
- → Capteur numérique

Sonde de température PT100 (capteur résistif avec Conv. Analog. Num. spécifique)
- → valeur numérique de la température

Sonde de température CTN
- → Capteur résistif non linéaire
- → Principe d'étalonnage d'un capteur

Capteurs U. V.
- → Option Sc et Labo : notion de protection de la peau

Multiples extensions possibles selon les projets ...

Propositions d'activités pour débutants ayant quelques notions du langage Python :

L'objectif ici est de mettre en application des notions déjà abordées en cours.

Notion de variable et d'affectation de valeur : var <- ... var = ...

Les tests conditionnels : == != < <= > >= si ... sinon si .... sinon ... if ... elif ... then ...

Introduction aux variables de type list *(indice des éléments présents dans une liste)* liste = [... , ...]

Boucle non bornée : tant que … while ... :

Boucle bornée : pour i allant de ... à ... for i in range(...) :

Préambule : consulter la fiche technique en annexe pour la mise en œuvre des fonctionnalités utilisées.

Activité 01	Si *condition est vraie* alors : ... Sinon : ...	if *condition* : ... else : ...

Afficher l'état du bouton poussoir (relâché ou enfoncé).

Activité 02	Tant que *condition est vraie* faire :	while *condition* :

Allumer la led verte.
Éteindre la led lorsque le bouton poussoir est enfoncé.

Exercice 1 en autonomie :
Allumer la led verte tant que le bouton poussoir est enfoncé.
Éteindre la led verte lorsqu'il est relâché

Activité 03-a	Tant que ... si ... compter ...	while *condition* / if ... / cptr = cptr + 1

Compter le nombre d'impulsions sur un bouton poussoir
→ Mise en évidence d'un dysfonctionnement par rapport à l'objectif à atteindre

Activité 03-b	Tant que ... si ... compter ...	while *condition* / if ... / cptr = cptr + 1

Compter le nombre d'impulsions sur le bouton poussoir
Correction du dysfonctionnement à l'aide d'une temporisation
→ Mise en évidence d'un nouveau dysfonctionnement par rapport à l'objectif à atteindre

Activité 03-c	Tant que ... si ... compter ...	while *condition* / if ... / cptr = cptr + 1

Compter le nombre d'impulsions sur un bouton poussoir
Correction avec la mémorisation du changement d'état
-> Logigramme : représentation des états en fonction du temps → algorithme

Activité 03-d	Appeler ma_fonction (...)	def ma_fonction (...) :

Déplacer une partie du code dans une fonction pour simplifier la lecture du code

Activité 04	Variable de type liste (tableau)	leds = [... , ... , ...]

Utiliser une variable de type list et accéder aux éléments de la liste en fonction de leur indice.

Exercice 2 en autonomie :
Utiliser la fonction randint(0,2) disponible dans la bibliothèque random pour allumer aléatoirement chacune des leds pendant une durée de 0.3 s et ceci jusqu'à ce que l'utilisateur appuie sur le bouton poussoir afin d'éteindre les leds.

```
1 from random import randint # importer la fonction randint
2 n = randint(0,2) # générer un nombre entier aléatoire compris entre 0 et 2 inclus.
```

Proposition de projet *(estimation 4h avec approfondissements à prévoir pour les élèves plus rapides) :*

Utilisation du ruban de huit leds NeoPixel *(led multicolor)*

Pour utiliser le ruban de huit leds NeoPixel, vous aurez besoin de la librairie neopixel dans laquelle est disponible la classe NeoPixel pour prendre en charge le fonctionnement du ruban.

```
1  from machine import Pin
2  from neopixel import NeoPixel
3
4  ruban = NeoPixel ( Pin(12), 8 ) # ruban de 8 leds neopixel connecté au GPIO 12
5  ruban.timing = (390, 800, 820, 430) # ajustement des temporisations → synchronisation des signaux.
```

Pour affecter une couleur à une led, on lui attribue un ensemble de trois nombres (tuple) pour chaque composante Rouge, Verte et Bleue de la lumière quelle va émettre.

```
6  ruban[0] = (80, 0, 80 )  # affecter une couleur magenta (rouge+bleu) à la première led de la liste (indice 0)
7  ruban.write( ) # envoyer l'information au ruban de leds.
```

Objectif 1 :

1 - Affecter une couleur différente à chaque led du ruban pour obtenir un dégradé de couleur : bleu → vert

2 - Éteindre le ruban de leds lorsqu'on appuie sur le bouton poussoir.

Objectif 2 :

1 - Déclarer une liste couleurs = [...] qui va contenir 10 combinaisons différentes de (R,V,B)

couleurs = [(... , ... , ...) , ...] *# liste des couleurs qui seront disponibles*

2 - Coder une boucle pour i allant de 0 à 7 inclus *# i → indice de la led neopixel*
affecter un nombre aléatoire compris entre 0 et 9 inclus à une variable n *# n → indice de la couleur*
affecter la couleur d'indice n à led neopixel d'indice i

3 - Envoyer l'information au ruban de leds neopixel

4 - Recommencer les étapes 2 et 3 lorsqu'on appuie une première fois sur le bouton poussoir

5 - Éteindre le ruban de leds lorsqu'on appuie une deuxième fois sur le bouton poussoir

Objectif 3 :

1 - Coder une fonction permuter_led_droite (...)
Cette fonction reçoit une variable ruban en argument *(ruban de leds neopixel)*
1 - Affecter la couleur de la led d'indice 0 à une variable memo.
2 - Pour i allant de 0 à 6 inclus :
La led d'indice i prend pour valeur la couleur de la led d'indice i+1
3 - La led d'indice 7 prend pour valeur la couleur affectée à la variable memo

2 - Déclarer une liste de 8 couleurs permettant d'obtenir un dégradé de couleurs du rouge au bleu

3 - Afficher le dégradé de couleur sur le ruban de leds neopixel

4 - Attendre 0,2 s

5 - Appeler la fonction permuter_led_droite (...) puis recommencer les étapes 3 et 4

6 - Changer le sens de permutation des leds lorsqu'on appuie une première fois sur le bouton poussoir

7 - Éteindre le ruban de leds lorsqu'on appuie une deuxième fois sur le bouton poussoir

Conseil : développer les étapes 1 à 5 et 7 dans un premier temps, puis intégrer l'étape 6 dans un deuxième temps.

Approfondissement possible : utiliser le mode interruption matérielle pour gérer les appuis sur le bouton poussoir (voir documentation en annexe).

Proposition de grille d'évaluation :

Nom(s) : Prénom(s) : Classe(s) :		*Interventions*	
		Algo.	**Python**
Objectif 1			
Déterminer huit 3_uplets (R,V,B) pour obtenir un dégradé de couleur			
Affecter une couleur différente à chacune des leds neopixels			
Coder un algorithme pour attendre et détecter l'appui sur le bouton poussoir			
Éteindre le ruban de leds neopixel (utilisation d'une boucle pour)			
Intervention pour apporter de l'aide			
Objectif 2			
Déclarer une liste de 3_uplets (R,V,B) pour les définir les 10 couleurs disponibles			
Déterminer un nombre n aléatoirement			
Coder une boucle pour ... afin d'affecter une couleur à chacune des 8 leds NeoPixel			
Affecter aléatoirement une couleur à chacune des leds NeoPixel			
Détecter l'appui sur le bouton poussoir *(prendre en charge les dysfonctionnements possibles)*			
Refaire l'affectation d'une couleur aléatoire à chaque led après l'appui sur le bouton			
Éteindre le ruban de leds après le deuxième appui sur le bouton poussoir			
Intervention pour apporter de l'aide			
Objectif 3			
Coder la fonction permuter_led_droite (...)			
Dégradé de couleurs du rouge au bleu			
Algorithme pour obtenir une permutation rotative à droite des leds			
Algorithme pour obtenir une permutation rotative à gauche des leds			
Gérer les deux appuis sur le bouton poussoir			
Éteindre le ruban de leds après deux appuis sur le bouton poussoir			
Intervention pour apporter de l'aide			
Conclusion			

Intervention régulière	Intervention régulière	Intervention régulière	Intervention régulière

Algorithmique

Des difficultés pour identifier les étapes d'un raisonnement afin de le traduire ensuite en algorithme.	Identifie correctement les étapes de la résolution mais rencontre des difficultés pour les traduire en algorithme	L'algorithme est quasi ou fonctionnel mais le choix des étapes pourrait être plus pertinent. *(fonctions, boucles, ...)*	Algorithme adapté et pertinent afin de résoudre le problème donnée.

Codage en langage Python

Manque de connaissances en langage Python. Progression difficile dans la traduction de l'algorithme. Des difficultés pour interpréter les messages d'erreur.	Des erreurs dans la syntaxe des instructions "simples" en langage Python. Doit approfondir la lecture et l'interprétation des messages d'erreur pour mieux identifier la nature du problème à corriger.	La syntaxe en Python est satisfaisante malgré quelques erreurs. Interprétation correcte des messages d'erreur pour corriger la syntaxe.	Bonne connaissance en langage Python. Autonomie satisfaisante pour la correction des bugs.

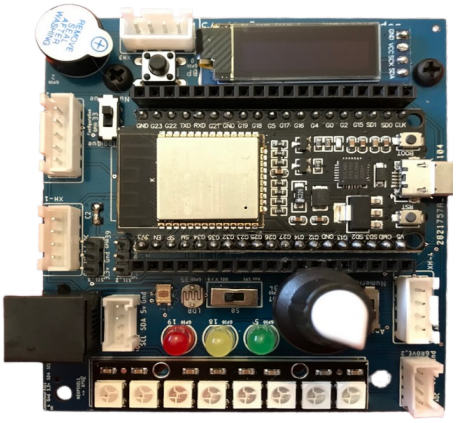
Annexe 1 : Mise en œuvre de quelques fonctionnalités

Utilisation des périphériques intégrés via les GPIO

(General Purpose Input / Output).

GPIO 19, 18, 5

Utilisation des **leds verte, jaune et rouge** :



```
1  from machine import Pin
2  from time import sleep
3
4  led_v = Pin( 5, Pin.OUT ) # led verte connectée au GPIO 5
5  led_j = Pin( 18, Pin.OUT ) # led jaune
6  led_r = Pin( 19, Pin.OUT ) # led rouge
7
8  led_V.on( ) # allumer la led verte ( → configurer le GPIO à l'état haut )
9  sleep( 2 )
10  led.off( )  # éteindre la led verte ( → configurer le GPIO à l'état bas )
```

GPIO 0

Utilisation du **bouton poussoir** en mode 'simple' :

Note : *Si le bouton est relâché, la valeur obtenue sera 1, s'il est enfoncé la valeur obtenue sera 0.*

```
1  from machine import Pin
2
3  bp = Pin( 0, Pin.IN ) # bouton poussoir connecté au GPIO 0
4
5  etat = bp.value( )  # lire l'état du bouton poussoir et affecter la valeur renvoyée à la variable etat
6
7  if etat == True :
8        print("Le bouton est relâché.")
9  else :
10       print("Le bouton est enfoncé.")
```

GPIO 12

Utilisation du **ruban de 8 leds NeoPixel**

```
1  from machine import Pin
2  from neopixel import NeoPixel
3  from time import sleep
4
5  ruban = NeoPixel( Pin(12), 8) # déclarer la variable ruban pour interagir avec les led NeoPixel
6  ruban.timing = (390, 800, 820, 430) # adaptation des temporisations → synchronisation des signaux
7
8  ruban[3] = ( 35, 35, 125) # Configurer la led d'indice 3 (la 4eme led du ruban) avec la couleur (35,35,125)
9  ruban.write( ) # envoyer l'information au ruban
10
11  sleep( 2 ) # attendre 2 s
12
13  for i in range(8) : # i prend pour valeurs successives :  0, 1, 2, 3, 4, 5, 6, 7
14     ruban[i] = (i*15, 105-i*15, 0) # configurer la couleur de chaque led : [ (0, 105, 0) , (15, 90, 0), ... ]
15  ruban.write( ) # envoyer la configuration au ruban
16
17  sleep( 1 )
18
19  for i in range(8) :
20     ruban[i] = (0, 0, 0) # aucune lumière → éteindre le ruban
21  ruban.write( )
```


GPIO 34 / GPIO 35 Utilisation du mode Conversion Analogique → Numérique (CAN)

Note : *Ce mode est utilisable pour le potentiomètre ou pour la LDR (Light Dependant Résistor)*

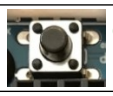
```
from machine import Pin ,ADC  # ADC → Analog to Digital Conversion ( CAN en français )

sensor = ADC ( Pin( 34 ) ) # configuration du GPIO 34 en mode ADC
sensor.atten ( ADC.ATTN_11DB ) # Atténuation de 11 dB → adaptation d'échelle si tension de 3,3V

val = sensor.read( )  # lire la valeur du CAN connecté au potentiomètre et l'affecter à la variable val
val_memo = val

print("Tournez le potentiomètre ...")
print("(Fin du script si la valeur obtenue est inférieure à 10) \n")  # \n → sauter une ligne
print("Valeur obtenue après mesure puis conversion : ", end="" ) # end="" → sur la même ligne
print("{:04d}".format(val), end="" ) # afficher la valeur sur 4 digits avec les zéro non significatifs

while val > 10 : # tant que la valeur obtenue est supérieure à 10
    val = sensor.read( ) # lire l'état du Conv. Analog. Numerique connecté au potentiomètre
    if val < val_memo-20 or val_memo+20 < val :
        val_memo = val
        print("\b"*4, end="") # effacer la précédente mesure (\b → backspace)
        print("{:04d}".format(val), end="" ) # mettre à jour l'affichage de la valeur
```



GPIO 0 Utilisation du **bouton poussoir** en mode **interruption matérielle** :

Plus d'infos : https://docs.micropython.org/en/latest/library/machine.Pin.html

```
""" L'utilisation des interruptions : irq (interrutpion request) permet d'éviter d'avoir à lire l'état
de la broche. La fonction (routine) de gestion de l'interruption va interagir avec le programme
principal (ici la routine affecte la valeur True à la variable globale bp_flag (drapeau)  )"""
from machine import Pin

def irq_flag ( pin ): # routine (fonction) de gestion de l'interruption
    global bp_flag # variable globale
    bp_flag = True # modification de la valeur affectée au drapeau

bp = Pin( 0 , Pin.IN )
bp.irq( trigger=Pin.IRQ_RISING, handler = irq_flag ) # utilisation de l'interruption sur front montant

bp_flag = False # variable globale (drapeau) aussi utilisée par la fonction de gestion d'interrutpion
while bp_flag == False :  # si appui sur le bouton : bp_flag <-True via la fonction irq_flag(...)
    pass  # ne rien faire
print("Ok pour interruption.")
```

GPIO 2 Utilisation du **buzzer** en mode PWM :

Plus d'infos : https://docs.micropython.org/en/latest/esp32/quickref.html#pwm-pulse-width-modulation

```
from machine import Pin, PWM
from time import sleep, sleep_ms # attente en secondes, sleep_ms → idem mais en millisecondes

buz = PWM ( Pin(2) ) # contrôle de la broche 2 en mode vibration : Pulse Wave Modulation

buz.freq( 440 ) # fréquence du signal : 440 Hz
sleep(1)
buz.duty(10) # diminuer l'intensité sonore à 10
sleep(1)
buz.duty(0) # intensité sonore à 0 → silence
sleep(1)

buz.deinit() # désactiver le buzzer
```