Seconde Option Sciences et Laboratoire Année 2019-2020

Objectif : **Réaliser un « radar de recul »**

Notions abordées lors de cette activité :

- Produire un son avec une fréquence déterminée :
  - → Calculer la période d'un son en microsecondes
  - → Contrôler l'état d'une broche

- Mesurer une distance avec un capteur à ultrason
  - → Confronter le résultat d'un calcul à une mesure
  - → Interpréter un résultat au regard des conditions expérimentales (température de l'air)

- Produire un son dont la fréquence et la durée sont déterminées en fonction de la distance où se situe l'obstacle (principe d'un radar de recul)

Principe :

Un radar de recul indique la présence d'un obstacle par un signal sonore.
La fréquence et la durée des sons émis doit donner une indication sur la distance de l'obstacle.

Séance N°1 :
Rédiger un algorithme puis le code Python afin de produire un son de fréquence déterminée.

Séance N°2 :
Rédiger un algorithme puis le code Python afin de mesurer une distance.
Calculer une moyenne pour prendre en compte la variation des mesures.
Rédiger un algorithme puis le code Python afin de donner une indications sur la distance mesurée à l'aide de signaux lumineux.

Séance N°3 :
Produire le code Python permettant de réaliser le radar de recul.
Aller plus loin :
- Rédiger une fonctionnement lorsque le code est répétitif
- Utiliser un indicateur lumineux graduel : huit diodes NEOPIXEL

Pour chaque séance, je demande de compléter un compte rendu de l'avancement des travaux :
→ Objectif atteint ou non
→ Difficultés rencontrées

Nom : Prénom : Date :

Séance N°1 :
Rédiger un algorithme puis le code Python afin de produire un son de fréquence variable.

Relation entre signal électrique et fréquence sonore

Lorsqu'un buzzer (ou Haut-Parleur) est relié à une broche, la membrane du buzzer se déplace dans un mouvement de va et vient selon l'état électrique de la broche 0 ou 1 (0V ou 5V).
Si on alterne l'état électrique de la broche on peut ainsi provoquer la vibration de la membrane et donc produire un son de même fréquence que la fréquence de changement de l'état électrique de la broche.

Rappel : La période T des changements d'état 0 - 1 permet de produire un son de fréquence F d'après la relation :	$F_{(Hz)} = \frac{1}{T_{(s)}}$	$T_{(s)} = \frac{1}{F_{(Hz)}}$

Produire un son de fréquence 440 Hz pendant 0,5s lorsqu'on appuie sur le bouton poussoir.

Etape N°1 : Contrôler l'état électrique d'une broche

1 - Lire la documentation sur le contrôle de l'état électrique d'une broche d'un microcontrôleur.
(Broche en mode Tout ou Rien : binaire)

2 - Coder un programme en Python qui correspond à l'algorithme ci-dessous :

Déclaration des broches (voir ci-contre) →

Boucler indéfiniment [`while True:`]
    si le bouton poussoir est enfoncé alors
        allumer la LED verte
        éteindre la broche LED rouge
    sinon :
        éteindre la LED Verte
        allumer la LED Rouge.

Prog00_BP_LED.py

```
1  from machine import Pin
2  LED_v = Pin ( 12, Pin.OUT )
3  LED_r = Pin ( 14, Pin.OUT )
4  Bouton = Pin ( 35, Pin.IN )
5
6  while True :
```

```
A vous d'écrire la suite ...
```

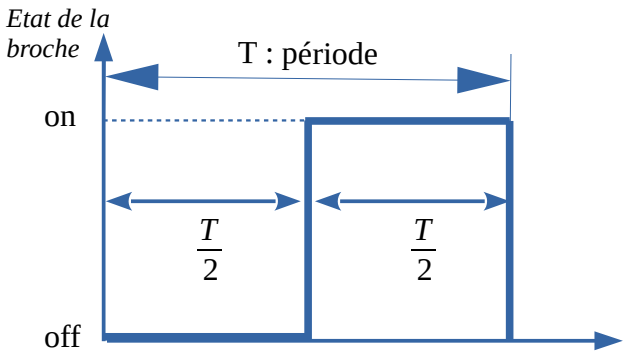
Pour info : *Après avoir cliqué dans la fenêtre [Console], la combinaison de touches* **[ctrl + c]** *permet d'arrêter l'exécution du programme en cours (par exemple dans le cas d'une boucle infinie [while True :])*

Etape N°2 : Algorithme pour contrôler la fréquence et la durée d'un son

Rédiger un algorithme qui permettrait de **produire un son de fréquence 440 Hz, pendant 0,5 seconde lorsqu'on appuie sur le bouton poussoir.**

Pour la production du son, on peut utiliser des fonctions d'attente en microsecondes pour obtenir une demie période : [Attendre n microsecondes]

Pour la durée du son il suffit de compter le nombre de périodes nécessaires (exemple : pour un son de 200 Hz il faut 200 périodes pour produire un son d'une seconde.
(Rappel : 200 Hz = 200 vibrations par seconde)



Etat de la broche pour produire une vibration.

Etape N°3 – Coder l'algorithme en MicroPython *(Réponse possible à l'étape N°2)*

Prog01_BP_BUZ_Algo.py *

```
# importer la configuration des broches
# importer la fonction sleep_us() de la bibliothèque [ time ]

# paramétrer la broche reliée au buzzer
# paraméter la broche reliée au bouton poussoir

# calculer la période en microseondes, d'un son de fréquence 440 Hz
# calculer le nombre de périodes pour obtenir une durée de 0,5s

# Faire une boucle infinie
    # lire l'état du bouton poussoir
    # Si le bouton poussoir est enfoncé alors
        # Faire une boucle pour le nombre de périodes nécessaires
            # Mettre la broche du buzzer à 0
            # Attendre une demie-période
            # Mettre la broche du buzzer à 1
            # Attendre une demie période
```

Prog02_BP_BUZ.py

```
from machine import Pin # importer la configuration des broches
from time import sleep_us # importer la fonction sleep_us()

Buzzer = Pin ( 25, Pin.OUT ) # Broche reliée au buzzer
Bouton = Pin ( 35, Pin.IN ) # Broche reliée au bouton poussoir

T = int( (1/440) * 10**6 ) # Période en microsecondes, d'un son de 440 Hz
print ( "Période : ", T, "µs" )
nbT = int ((0.5 * 10**6 ) / T) # nombre de périodes pour une durée de 0,5 s
print ( "Nombre de périodes : ", nbT )

while True: # boucle infinie
```

Etape N°4 – Pour aller plus loin ...

Si vous êtes joueur … on peut passer au niveau supérieur ;-)

Objectif : Produire un son dont la fréquence varie de 200 Hz à 4295 Hz selon la position du potentiomètre). Le programme s'arrête lorsqu'on appuie sur le bouton poussoir.

(voir notice Contrôler l'état électrique d'une broche // Broche en mode analogique)

BILAN

Les objectifs ont-ils été atteints ?	Quelles ont été les difficultés rencontrées ?

CONTROLER L'ETAT ÉLECTRIQUE D'UNE BROCHE D'UN MICROCONTRÔLEUR.

Pour contrôler l'état électrique des broches (en anglais 'Pin') , on dispose des fonctions [`Pin`] et [`ADC`] contenues dans la bibliothèque [`machine`].

Ces fonctions peuvent être utilisées après les avoir importées en mémoire :
[`from machine import Pin`] → utilisation en TOUT ou RIEN (Binaire : 1 ou 0 // Vrai ou Faux)
[`from machine import ADC`] → utilisation en mode NUMÉRIQUE (valeur comprise entre 0 et 4095)
ADC : Analog to **D**igital **C**onversion

Broche en mode TOUT [`1`] ou RIEN [`0`] → Mode binaire

Principe d'utilisation de la fonction Pin : [*nom_objet* **= Pin (** *N°de broche* **, Pin.IN** *ou* **Pin.OUT)]**

Broche en entrée : **Pin.IN**

Une broche peut soit être utilisée pour lire un état électrique (ex : le signal d'un capteur).
Elle est alors dite broche en entrée : Pin.IN

[`bouton=Pin(35, Pin.IN)`] permet de connaître l'état du bouton poussoir relié à la broche 35 :
[`etat = bouton.value()`] → Lecture de l'état du bouton
- si le bouton est enfoncé, [`etat`] a pour valeur « `1` »
- si le bouton est relâché, [`etat`] a pour valeur « `0` ».

Broche en sortie : **Pin.OUT**

Au contraire, pour contrôler un périphérique (actionneur, moteur, buzzer, …) la broche doit être configurée en sortie : Pin.OUT.

[`ledv=Pin(12, Pin.OUT)`] permet de contrôler l'état de la LED verte reliée à la broche 12 :
- [`ledv.on( )`] fait passer la broche à l'état HAUT et donc d'allumer la LED verte
- [`ledv.off( )`] fait passer la broche à l'état BAS et donc d'éteindre la LED verte

Broche en mode analogique : valeur comprise entre 0 et 4095

Principe d'utilisation de la fonction ADC : [*nom_objet* **= ADC (** *N°de broche* **)]**

Certaines broches permettent de mesurer un état électrique variable. L'état de la broche peut alors prendre des valeurs comprises entre 0 et 4095.

Exemple : si on souhaite connaître la position du potentiomètre, on peut lire l'état de la broche N°34.

```
from machine import Pin, ADC
from time import sleep_ms

bouton_pot = ADC ( Pin(34) )        # créer une broche analogique
bouton_pot.atten(ADC.ATTN_11DB)     # adaptation des mesures et valeurs

while True :
     valeur = bouton_pot.read()     # lecture de l'état de la broche
     print ('Valeur mesurée : ', val ) # afficher la valeur mesurée
     sleep_ms(200)                  # pause de 200 ms
```

Nom : Prénom : Date :

Séance N°2 :
Rédiger un algorithme puis le code Python afin de mesurer une distance
Rédiger un algorithme pour calculer la moyenne d'une série de mesures
Rédiger un algorithme puis le code Python afin de réagir différemment selon la distance mesurée.

Etape N°1 : Mesurer une distance à l'aide d'un capteur à ultrasons

Importer la bibliothèque SOPROLAB_UltraSonV1.py enregistrée dans le microcontrôleur.

Cette bibliothèque vous permet ensuite de mesurer une distance en millimètres.

Vous pouvez tester le bon fonctionnement du capteur par ces quelques lignes.

Note :
Le microcontrôleur ne doit pas être sous tension lorsque vous branchez le capteur de distance !

Prog04_HCSR.py

```
1  from SOPROLAB_UltraSonV1 import *
2
3  d = HCSR.distance_mm()
4  print("La distance est de ", d, "mm")
```

Console

```
>>> %Run -c $EDITOR_CONTENT
  La distance est de  1109 mm
>>>
```

Etape N°2 : Calculer une valeur moyenne

A partir de l'algorithme ci-dessous, **rédiger le programme en langage Python** qui permet d'effectuer une série de 5 mesures espacées de 0,5s et de calculer la moyenne des valeurs mesurées**.**

Prog05_HCSR_moy_algo.py

```
1  from time import sleep_ms # importer la fonction d'attente en millisecondes
2  from SOPROLAB_UltraSonV1 import * # importer la bibliothèque pour gérer le capteur
3                                    # de distance à ultrasons
4
5  # Faire 5 mesures
6      # Mesurer la distance
7      # Afficher la mesure obtenue
8      # Calculer la somme des mesures
9      # Attendre 500 ms
10 # Diviser la somme des mesures par 5
11 # Afficher le résultat
```

Etape N°3 – Mesure de distance et indicateur lumineux

1 - Ecrire l'algorithme d'un programme qui permet d'obtenir une indication sur la distance à laquelle est situé un obstacle.
Le programme devra :

- s'arrêter lorsqu'on appuie sur le bouton poussoir,
- allumer que la LED verte si l'obstacle est situé à plus d'un mètre,
- allumer les deux LED verte et jaune si l'obstacle est situé entre un mètre et 50 cm,
- allumer les trois LED verte, jaune et rouge si l'obstacle est à moins de 50 cm,
- effectuer une pause de 200 ms entre deux mesures.

2 - Rédiger le programme en langage Python qui correspond à l'algorithme

Prog08_HCSR_LED.py

```
1  from machine import Pin # importer la configuration des broches
2  from time import sleep_ms # importer la fonction d'attente en millisecondes
3  from SOPROLAB_UltraSonV1 import * # bibliothèque du capteur de distance à ultrasons
4
5  ledv = Pin ( 12, Pin.OUT) # Broche reliée à la LED verte
6  ledj = Pin ( 13, Pin.OUT) # Broche reliée à la LED jaune
7  ledr = Pin ( 14, Pin.OUT) # Broche reliée à la LED rouge
8  Bouton = Pin ( 35, Pin.IN ) # Broche reliée au bouton poussoir
9
```

à vous d'écrire la suite ...

Etape N°4 – Ondes radar et furtivité !

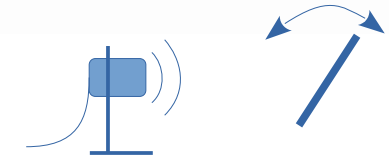
Afficher les mesures de distance en continu tant que le bouton poussoir n'a pas été enfoncé :

Prog08_HCSR_Furtif.py

```
from machine import Pin # importer la configuration des broches
from time import sleep_ms # importer la fonction d'attente en millisecondes
from SOPROLAB_UltraSonV1 import * # bibliothèque du capteur de distance à ultrasons

Bouton = Pin ( 35, Pin.IN ) # Broche reliée au bouton poussoir

while Bouton.value() == False: # Tant que le bouton poussoir n'a pas été enfoncé
    d = HCSR.distance_mm( )    # mesurer la distance
    print("Distance mesurée :", d, "mm") # afficher le résultat de la mesure
    sleep_ms(300) # temporisation entre deux mesures
```



Placer devant le capteur une plaque lisse et modifier l'orientation de la plaque face au capteur de distance à ultrason :

Comment pouvez-vous interpréter les résultats affichés ?

Les ondes radar fonctionnent comme le capteur à ultrasons. Proposer alors une explication aux formes particulières du bombardier F-117 aussi appelé bombardier furtif car difficilement repérable par les radars (Cf photo ci-contre) →

Etape N°5 – Pour préparer la prochaine séance :

On a pour objectif de **produire un son dont la fréquence (ou la période) et la durée dépendent de la distance à** laquelle se situe un obstacle (principe du radar de recul).

Informations : la période et la durée du son émis seront déterminées d'après les relations suivantes :

Période (µs) = [centre de l'intervalle qui contient la distance mesurée (en mm)] * 5 + 220
Durée du son (ms) = Période (µs) * 0,4
Temps d'attente ente deux mesures est égal à : la durée du son / 2.

Déterminer les caractéristiques des sons à émettre selon la distance de l'obstacle.
Vous pouvez utiliser une feuille de calculs sous LibreOffice pour compléter ce tableau.

Intervalles de distance (mm)	0 ≤ …< 100	100 ≤ …< 200	200 ≤ …< 300	300 ≤ …< 400	400 ≤ …< 500	500 ≤ ...
Centre de l'intervalle						Aucun son
Période (µs)						
Durée (ms)						
Nombre de périodes						

BILAN

Les objectifs ont-ils été atteints ?	Quelles ont été les difficultés rencontrées ?

Nom : Prénom : Date :

Séance N°3 :
Réinvestir les travaux des deux précédentes séances
Rédiger un algorithme puis le code Python afin de produire une indication sonore en fonction de la distance mesurée.

Etape N°1 : Algorithme du projet

Rédiger l'algorithme du programme selon lequel, le microcontrôleur doit :

- S'arrêter dès qu'on appuie sur le bouton poussoir,
- Mesurer la distance à laquelle se situe l'obstacle,
- Emettre un son dont la fréquence (période) et la durée dépendent de la distance mesurée :

Intervalles de distance (mm)	0 ≤ …< 100	100 ≤ …< 200	200 ≤ …< 300	300 ≤ …< 400	400 ≤ …< 500	500 ≤ ...
Centre de l'intervalle	*50*	*150*	*250*	*350*	*450*	Aucun son
Période (µs)	470	970	1470	1970	2470	
Durée (ms)	188	388	588	788	988	
Nombre de périodes	400	400	400	400	400	

Prog9_HCSR_BUZ_algo.py

```
# importer la configuration des broches
# importer la fonction d'attente en microsecondes
# importer la bibliothèque pour gérer le capteur de distance à ultrasons

# paramétrer la broche reliée au buzzer
# paraméter la broche reliée au bouton poussoir

# Par défaut on fixe la période à 2500µs (obstacle au delà de 500mm)

# Tant que le bouton poussoir n'a pas été enfoncé
```

à vous d'écrire la suite …

Etape N°2 : Coder la version finale du projet en langage Python

Prog9_HCSR_BUZ.py

```
from machine import Pin # importer la configuration des broches
from time import sleep_us # importer la fonction d'attente en microsecondes
from SOPROLAB_UltraSonV1 import * # bibliothèque du capteur de distance à ultrasons

Buzzer = Pin ( 25, Pin.OUT ) # Broche reliée au buzzer
Bouton = Pin ( 35, Pin.IN ) # Broche reliée au bouton poussoir

periode = 2500 # Par défaut période = 2500µs (obstacle au delà de 500mm)

while Bouton.value() == False: # Tant que le bouton poussoir n'a pas été enfoncé
```

À vous de coder la suite … (le microcontrôleur devra faire une pause de 10ms minimum entre deux mesures)

Etape N°3 : Pour aller plus loin ...

On constate que la production du son est un code répétitif, On peut donc **rédiger une fonction** en Python pour produire les différents sons.

Cette fonction prendra comme arguments la période en microsecondes :

```
def Produire_son ( periode ):
    demie_periode = periode // 2
    for cptr in range(400):
```

à vous de coder la suite ...

Etape N°4 : Pour aller encore plus loin ...

Vous pouvez utiliser les huit LED NeoPixel pour fournir une indication lumineuse avec un dégradé du vert (obstacle au-delà de 500mm) jusqu'au rouge (obstacle inférieur à 100mm).

NEOPIXEL [n] = (niv ROUGE, niv VERT, niv BLEU) avec n ∈ [0 ; 7]

Distance (mm)	… < 100	100 - 200	200 - 300	300 - 400	400 - 500	500 - 600	600 - 700	Au delà
NEOPIX[0]	(0 , 140 , 0)							
NEOPIX[1]	(20, 120, 0)							off
NEOPIX[2]	(40 , 100, 0)						off	
NEOPIX[3]	(60, 80, 0)					off		
NEOPIX[4]	(80, 60, 0)				off			
NEOPIX[5]	(100, 40, 0)			off				
NEOPIX[6]	(120, 20, 0)		off					
NEOPIX[7]	(140, 0, 0)	off						

Rédiger l'algorithme puis le code Python pour utiliser les LED NeoPixel.

Prog10_HCSR_NEOPIX_algo.py

```
# importer la configuration des broches
# importer la fonction d'attente en millisecondes
# importer la bibliothèque pour gérer le capteur de distance à ultrasons
# importer la bibliothèque pour gérer les diodes NEOPIXEL

# paramétrer la broche reliée au buzzer
# paraméter la broche reliée au bouton poussoir

# Configurer la diode NEOPIXEL[0] avec ( 0, 140, 0 )

# Tant que le bouton poussoir n'a pas été enfoncé
```

à vous d'écrire la suite …

Prog10_HCSR_NEOPIX.py

```
from machine import Pin # importer la configuration des broches
from time import sleep_ms # importer la fonction d'attente en millisecondes
from SOPROLAB_UltraSonV1 import * # bibliothèque du capteur de distance à ultrasons
from SOPROLAB import * # importer la bibliothèque pour gérer les diodes NEOPIXEL

Bouton = Pin ( 35, Pin.IN ) # Broche reliée au bouton poussoir

NEOPIX[0] = (0, 140, 0) # Configurer la diode NEOPIXEL[0] avec ( 0, 140, 0 )

while Bouton.value() == False: # Tant que le bouton poussoir n'a pas été enfoncé
```

à vous d'écrire la suite …

Etape N°5 : Pour aller toujours plus loin ...

De même que pour l'étape n°3, lorsque le code est répétitif, on peut le simplifier avec une fonction.

Vous pouvez constater que la composante rouge augmente de 20 en 20 alors que la composante verte, elle, diminue de 20 en 20. La somme des deux fait donc toujours 140 …

En attribuant un indice à chaque intervalle de distance : 1, 2, 3, 4 … vous pouvez calculer plus facilement les caractéristiques pour chaque diode NEOPIXEL :

NEOPIXEL [0] → toujours (0, 140, 0)
Pour i dans l'intervalle [1 ; 7] :
NEOPIXEL [i] = (20×i, 140 – 20×i , 0)