2. Le protocole de communication "Firmata Express"

Le protocole de communication "Firmata Express" est une version améliorée de la bibliothèque "Firmata Standard 2.5.8" pour Arduino. Il a été conçu pour être utilisé conjointement avec le client Firmata "pymata-express" dans les programmes en Python.

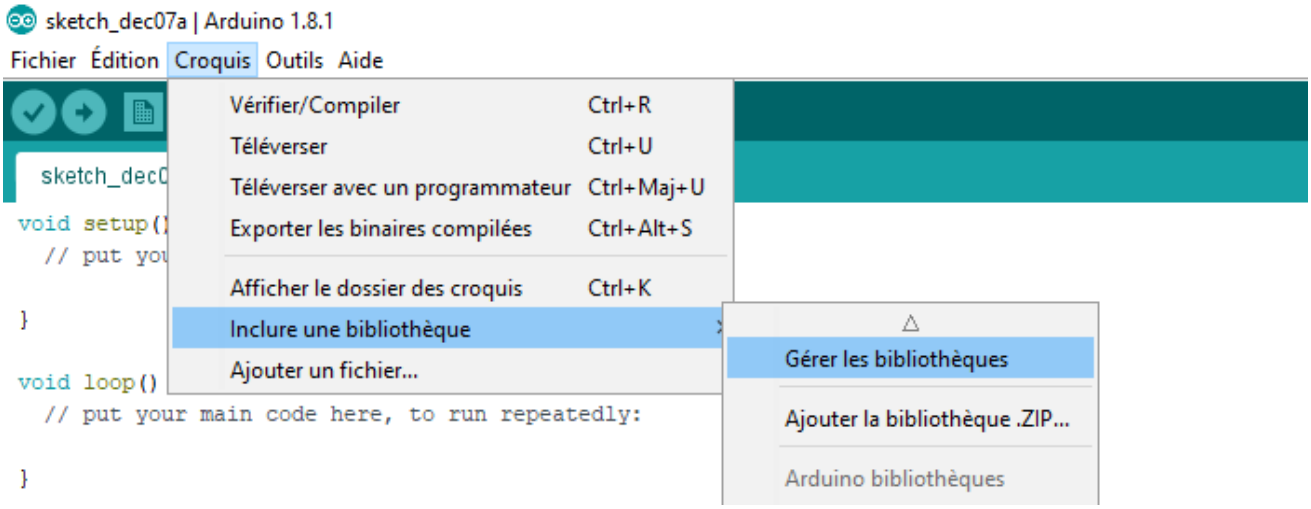
Il prend en charge toutes les fonctionnalités de la bibliothèque "Firmata Standard 2.5.8" et ajoute en outre la prise en charge de :

- La bibliothèque Arduino "Tone"
- Capteurs de distance HC-SR04
- Moteurs pas à pas

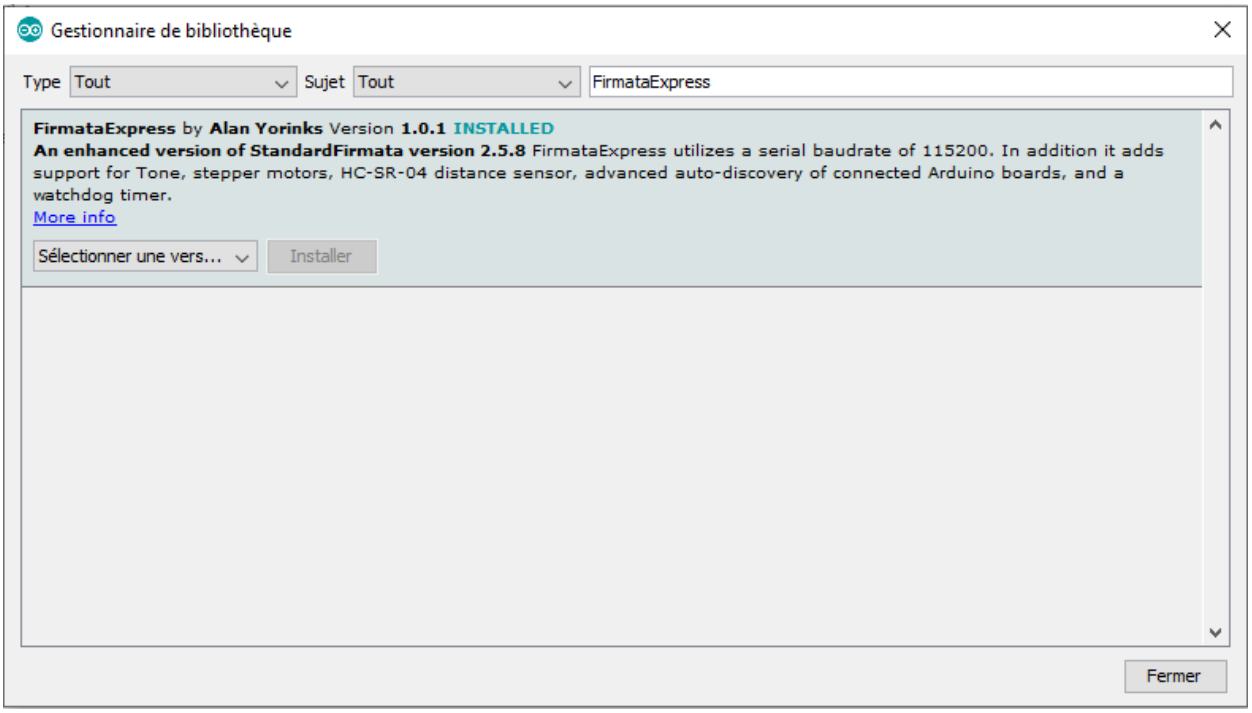
La liaison série fonctionne à un débit de 115200 bauds, soit deux fois la vitesse de "Firmata Standard".

. **Chargement du code "Firmata Express"** :

- Brancher l'Arduino via un port USB,
- Ouvrir le logiciel "**IDE ARDUINO**",
- Sélectionner "**Croquis/Inclure une bibliothèque/Gérer les bibliothèques**",



- Entrer "**FirmataExpress**" dans la zone de recherche :



- et cliquer sur "**installer**".

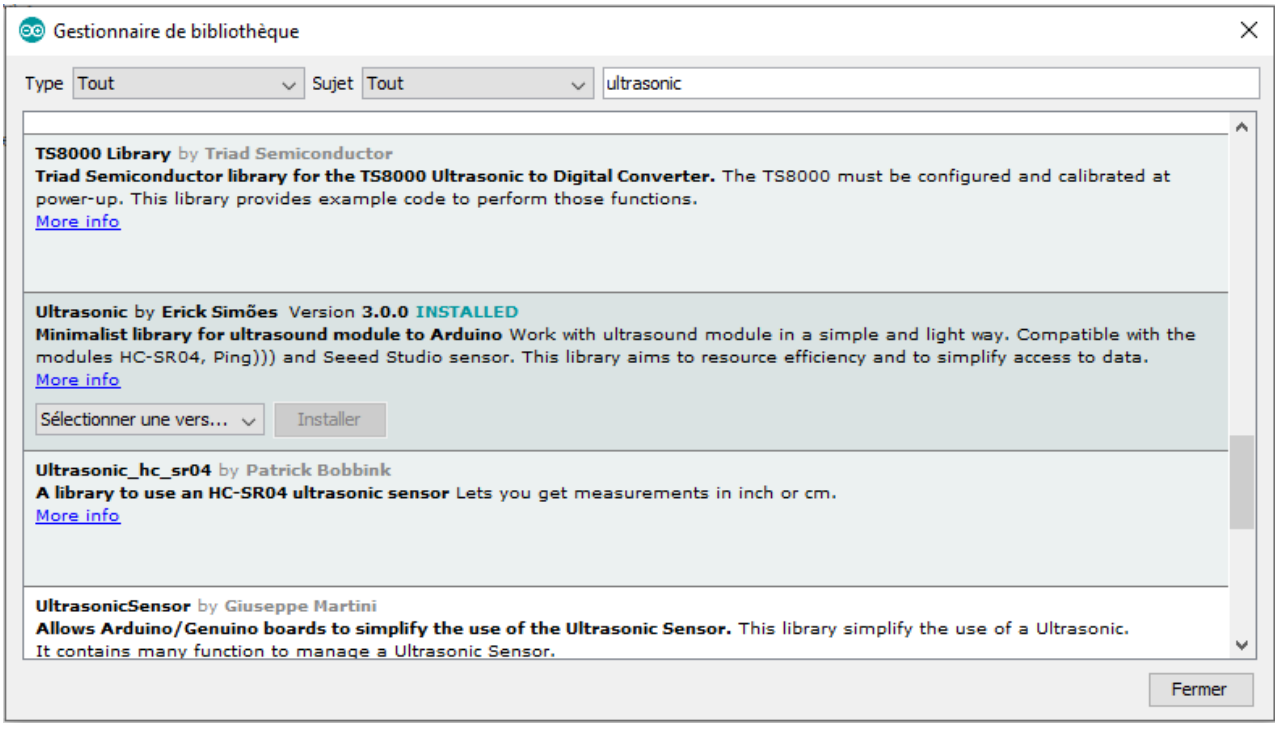
Firmata Express nécessite également que la librairie "Ultrasonic by Erick Simões" soit installée.

De même que précédemment, en utilisant le logiciel Arduino IDE :

- Sélectionner "**Croquis/Inclure une bibliothèque/Gérer les bibliothèques**",

- Entrer "**Ultrasonic**" dans la zone de recherche,

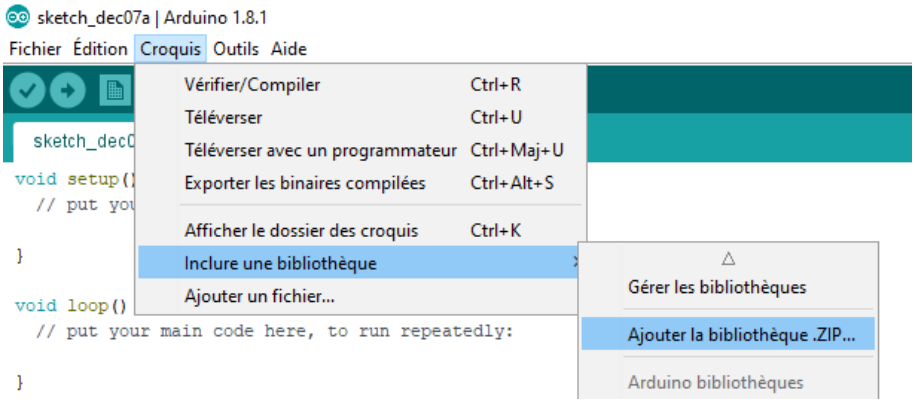
- Sélectionner "**Ultrasonic by Erick Simões**"



- et cliquer sur "**installer**".

Les librairies peuvent aussi être installées à partir des fichiers "**zip**" présents dans le dossier "**Support/Librairies Arduino**" :

- Ouvrir le logiciel "**IDE ARDUINO**",
- Sélectionner "**Croquis/Inclure une bibliothèque/Ajouter la bibliothèque .ZIP**",
- Sélectionner le fichier .ZIP de la librairie à installer.



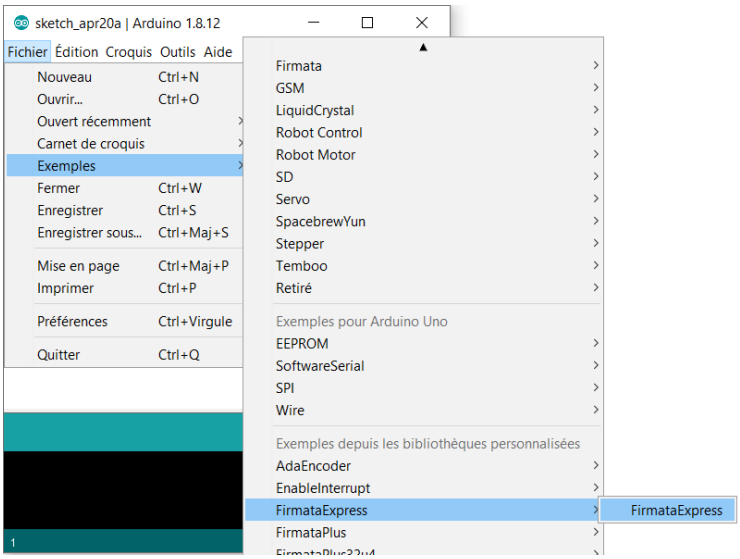
Ou tout simplement en copiant le contenu du dossier "**Support/libraries**" dans le dossier des librairies du logiciel "**Arduino IDE**" (sous windows, ce dossier se trouve dans le dossier **"documents/Arduino"**).

. <u>Chargement du code "Firmata Express" dans la mémoire de l'Arduino</u> :

- Brancher l'Arduino via un port USB,
- Afin de charger la librairie "Firmata express" sur l'ARDUINO, il faut lancer le logiciel "IDE ARDUINO", puis sélectionner :

Fichier > Exemples > FirmataExpress > FirmataExpress,

- puis cliquer sur "téléverser".



. <u>Installation de la bibliothèque "pymata-express" dans Python:</u>

La bibliothèque "**pymata-express**" n'est compatible qu'avec la version 3.7 de Python au minimum.
Pour faire fonctionner, un programme en Python qui contrôle l'Arduino via le protocole de communication **"Firmata Express"**, Python doit disposer de la bibliothèque "**pymata-express**". Celle-ci peut être installée via **"pip"**, à l'aide de la ligne de commande :

pip install pymata-express==1.4

La ligne de commande d'installation de "**pymata-express**" précise la version de la bibibliothèque à installer.
Il est important d'installer la version **1.4** de "**pymata-express**" pour assurer le bon fonctionnement des programmes avec les capteurs ultrasoniques.

Pour utiliser la bibliothèque **"pymata-express"** dans un programme python, il faut importer le module "**PymataExpress**", à l'aide de l'instruction :

from pymata_express.pymata_express import PymataExpress

La connexion avec le microcontrôleur, via le port série, est réalisée avec ce module en précisant le port COM sur lequel l'Arduino est connecté :

board = PymataExpress(com_port = PortComArduino)

Une fois la connexion établie, il est possible d'interroger ou de modifier les entrées et sorties numériques ou analogiques de l'Arduino.

"pymata-express" utilise la bibliothèque "asyncio Python 3.7" pour la prise en charge de la programmation asynchrone (programme exécutant plusieurs codes simultanément de façon non bloquante) en tirant parti des tâches "asyncio" (ou coroutines).

Tous les actions demandées à l'Arduino seront transmises par l'intermédiaire de tâches "asyncio" à définir.

2.1 Gestion des sorties numériques

Pour modifier l'état d'une sortie numérique (l'équivalent d'un "digitalWrite" en langage Arduino), il faut au préalable déclarer la broche en sortie digitale à l'aide de l'instruction suivante :

loop.run_until_complete(board.set_pin_mode_digital_output(pin))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express** ",
- "**pin**" est le numéro de la broche du microcontrôleur que l'on souhaite déclarer en sortie digitale,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

Le plus simple est de créer une fonction qui sera appelée dans nos programmes en Python pour déclarer une broche en sortie numérique :

```
def Set_DigitalOutput_Pin(board, pin):
  loop.run_until_complete(board.set_pin_mode_digital_output(pin))
```

Ainsi, l'instruction pour déclarer la broche "11", de l'objet "board", en sortie numérique est :

Set_DigitalOutput_Pin(board, 11)

L'état logique de la sortie est alors modifié à l'aide de l'instruction suivante :

loop.run_until_complete(board.digital_write(pin, val))

De même, on peut définir une fonction pour l'écriture sur une sortie digitale :

```
def Digital_Write(board, pin, val):
  loop.run_until_complete(board.digital_write(pin, val))
```

Ainsi, l'instruction pour mettre la sortie numérique "11", de l'objet "board", à l'état haut est:

Digital_Write(board, 11, 1)

Exemple :

L'exemple d'application de la gestion des sorties numériques avec "Firmata Express" est le même que celui avec "Firmata Standard".
Le programme adapté à la bibliothèque "**pymata_express**" demande à l'utilisateur de saisir une instruction (si l'instruction est "ON", la DEL rouge s'allume, et si c'est "OFF", la DEL rouge s'éteint (programme "LedRVBDigitalOutput.py" dans le dossier "Codes/PY/ FirmataExpress").

```
# Importations des librairies et définition de fonctions

from ConnectToArduino import *
import asyncio

def Set_DigitalOutput_Pin(board, pin):
    loop.run_until_complete(board.set_pin_mode_digital_output(pin))

def Digital_Write(board, pin, val):
    loop.run_until_complete(board.digital_write(pin, val))

def Arduino_Exit(board):
    loop.run_until_complete(board.shutdown())


# Déclaration des constantes et variables

PinLedR = 11


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

loop = asyncio.get_event_loop()
Set_DigitalOutput_Pin(board, PinLedR)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter\n")


# Boucle principale du programme

print("Pour allumer la DEL rouge, envoyer: ON");
print("Pour éteindre la DEL rouge, envoyer: OFF\n");

while True:
    try:
        saisie = str(input(""))
        if saisie=="ON" : Digital_Write(board,PinLedR,1)
        elif saisie=="OFF" : Digital_Write(board,PinLedR,0)

    except KeyboardInterrupt:
        Digital_Write(board,PinLedR,0)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module ”**ConnectToArduino.py**”, contenant les fonctions de connexion à l’arduino via le protocole ”**Firmata Express**”, est importé. L’importation de ce module importe également les bibliothèques :

- ”**PymataExpress**” pour contrôler l’Arduino,
– ”**serial.tools.list_ports**” pour déterminer la liste des ports COM disponibles,
– ”**sys**” pour sortir du programme.

. La bibliothèque ”**asyncio**” nécessaire au fonctionnement de ”**PymataExpress**”

. Fonction **" Set_DigitalOutput_Pin "** pour déclarer une broche en sortie numérique,

. Fonction **"Digital_Write"** pour modifier l’état logique d’une broche numérique,

. Fonction **"Arduino_Exit"** pour fermer le port COM et se déconnecter de l’Arduino

- Déclaration des constantes et variables :

. **PinLedR = 11** (constante correspondant au n° de la broche sur laquelle la DEL rouge est connectée)

. **PortComArduino** (port COM sur lequel l’Arduino est connecté)

- Connexion à l'Arduino :

. Appel de la fonction de sélection du port COM du module ”**ConnectToArduino.py**”:

PortComArduino = SelectPortCOM()

Le nombre de port COM disponible est alors déterminé :

PortsCOM = list(serial.tools.list_ports.comports()

–> si nombre de port COM = 0 : message d’erreur,

–> si nombre de port COM = 1 : sélection de ce port COM pour la connexion,

–> si nombre de port COM > 1 : L’utilisateur doit sélectionner le bon port COM.

```
def SelectPortCOM():

    Nport=[]
    PortsCOM = list(serial.tools.list_ports.comports())
    for port_numero, description, address in PortsCOM:
        Nport.append(port_numero)


    if len(PortsCOM)== 0:
        print("Aucune carte Arduino n'a été détectée!")
        saisie = ""
        while saisie != "q":
            saisie = str(input("Entrez 'q' pour quitter: "))
        sys.exit()

    elif len(PortsCOM)== 1:
        PortComArduino = Nport[0]

    else:
        print("Liste des ports COM disponibles:\n")
        for i in range(len(PortsCOM)):
            print(i+1, ": ", PortsCOM[i])
        ChoixPort=False
        while ChoixPort==False:
            Choix = input("\nVeuillez indiquer le numéro du port de la carte Arduino:")
            try:
                Choix = int(Choix)
                assert Choix >= 1 and Choix <= len(PortsCOM)
                ChoixPort = True
            except AssertionError:
                print("Le numéro indiqué n'est pas entre 1 et", len(PortsCOM) ,"!")
                ChoixPort = False
            except:
                print("Vous n'avez pas saisi un numéro entre 1 et", len(PortsCOM) ,"!")
        PortComArduino = Nport[Choix-1]

    return PortComArduino
```

. Tentative d'ouverture du port COM sélectionné (PortComArduino) et de connexion à l'Arduino via le protocole **"Firmata Express"** avec la fonction **"OpenPortCom"** du module **"ConnectToArduino.py"** :

board = OpenPortCom(PortComArduino)

```
def OpenPortCom(PortCom):

    try:
        board = PymataExpress(com_port = PortCom)

    except:
        AffichMessageErreur(PortCom)

    else:
        return board


def AffichMessageErreur(PortCom):

    print("Un problème s'est produit à l'ouverture du port.\n"
        "Vérifiez que le port utilisé par la carte Arduino est bien "+ PortCom + ",\n" +
        "et que le protocole de communication FIRMATA EXPRESS a bien été téléversé.\n")
    saisie = ""
    while saisie != "q":
        saisie = str(input("Entrez 'q' pour quitter: "))
    sys.exit()
```

–> message d'erreur et le programme est arrêté si l'ouverture du port COM sélectionné ou la création de l'objet **"board"** a échoué.

Un échec lors de la connexion se produit si le port COM indiqué ne correspond pas à un port Arduino ou si le protocole "Firmata Express" n'a pas été téléversé dans la mémoire de l'Arduino.

. Si la connexion à l'Arduino est réussie:

→ définition d'une boucle asyncio : **loop = asyncio.get_event_loop()**

→ Déclaration de la broche de la DEL rouge en sortie numérique :

Set_DigitalOutput_Pin(board, PinLedR)

- Boucle principale du programme (boucle "while True") :

. Attente de la saisie de l'instruction :

saisie = str(input(""))

. La DEL rouge est allumée ou éteinte suivant l'instruction saisie :

if saisie=="ON" : Digital_Write(board,PinLedR,1)

elif saisie=="OFF" : Digital_Write(board,PinLedR,0)

- Fin du programme en appuyant sur Ctrl-C :

→ La DEL rouge est éteinte et le port série est fermé.

2.2 Gestion des entrées numériques

Pour lire l'état logique d'une broche numérique (par exemple, la broche N°12), il faut la déclarer au préalable en entrée avec la commande suivante :

loop.run_until_complete(board.set_pin_mode_analog_input(12))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express** ",
- "**12**" est le numéro de la broche du microcontrôleur que l'on souhaite déclarer en entrée digitale,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

On peut définir une fonction déclarant plus facilement une broche en entrée numérique :

```
def Set_DigitalInput_Pin(board, pin):
   loop.run_until_complete(board.set_pin_mode_digital_input(pin))
```

La syntaxe pour déclarer la broche N°12 en entrée numérique est alors plus simple :

Set_DigitalInput_Pin(board,12)

Ensuite on pourra lire l'état logique de la broche au moyen de cette instruction :

value = loop.run_until_complete(board.digital_read(12))

qui retourne une liste dont le premier élément (**value[0])** est égale à "**1**" lorsque l'entrée est à **5 V**, et "**0**" lorsqu'elle est à **0 V**.
De même, on peut définir une fonction pour lire l'état logique d'une entrée digitale :

```
def Digital_Read(board, pin):
   value = loop.run_until_complete(board.digital_read(pin))
   return value[0]
```

Ainsi, l'instruction pour lire l'état logique de la broche N°12 devient :

ValPin12 = Digital_Read(board, 12)

Exemple :

L'exemple d'application de la gestion des entrées numériques avec "Firmata Express" est le même que celui avec "Firmata Standard".

Le programme suivant (programme "LedRVBDigitalInput.py" dans le dossier "Codes/PY/FirmataExpress") permet d'allumer la DEL rouge quand le bouton poussoir est appuyé et de l'éteindre quand celui-ci est relâché :

```
# Importations des librairies et définition de fonctions

from ConnectToArduino import *
import asyncio


def Set_DigitalInput_Pin(board, pin):
    loop.run_until_complete(board.set_pin_mode_digital_input(pin))

def Digital_Read(board, pin):
    value = loop.run_until_complete(board.digital_read(pin))
    return value[0]

def Set_DigitalOutput_Pin(board, pin):
    loop.run_until_complete(board.set_pin_mode_digital_output(pin))

def Digital_Write(board, pin, val):
    loop.run_until_complete(board.digital_write(pin, val))

def Arduino_Exit(board):
    loop.run_until_complete(board.shutdown())

# Déclaration des constantes et variables

PinLedR = 11
PinButton = 12
ValButton = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

loop = asyncio.get_event_loop()
Set_DigitalOutput_Pin(board, PinLedR)
Set_DigitalInput_Pin(board, PinButton)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter\n")


# Boucle principale du programme
print("Appuyez sur le bouton poussoir pour allumer la DEL.")

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        if ValButton == 1:
            Digital_Write(board,PinLedR,1)
        else:
            Digital_Write(board,PinLedR,0)

    except KeyboardInterrupt:
        Digital_Write(board,PinLedR,0)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module "**ConnectToArduino.py**", contenant les fonctions de connexion à l'arduino via le protocole "Firmata Express",
. La bibliothèque **"asyncio"** nécessaire au fonctionnement de **"PymataExpress"**
. Fonction **" Set_DigitalOutput_Pin "** pour déclarer une broche en sortie numérique,
. Fonction **"Digital_Write"** pour modifier l'état logique d'une broche numérique,
. Fonction **" Set_DigitalInput_Pin "** pour déclarer une broche en entrée numérique,
. Fonction **"Digital_Read"** pour lire l'état logique d'une broche numérique,
. Fonction **"Arduino_Exit"** pour fermer le port COM et se déconnecter de l'Arduino.

- Déclaration des constantes et variables :

. **PinLedR = 11** (constante correspondant au n° de la broche sur laquelle la DEL rouge est connectée)
. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)
. **ValButton = 0** (variable pour stocker la valeur de l'état logique de la broche du bouton poussoir)
. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

. Tentative d'ouverture du port COM sélectionné et connexion à l'Arduino:

```
PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)
```

. Si la connexion à l'Arduino est réussie:

→ définition d'une boucle asyncio : **loop = asyncio.get_event_loop()**

→ Déclaration de la broche de la DEL rouge en sortie numérique :
Set_DigitalOutput_Pin(board, PinLedR)

→ Déclaration de la broche du bouton poussoir en entrée numérique :
Set_DigitalInput_Pin(board, Pinbutton)

- Boucle principale du programme (boucle "while True") :

. Lecture de l'état logique de la broche du bouton poussoir :

```
ValButton = Digital_Read(board, PinButton)
```

. La DEL rouge est allumée ou éteinte suivant la valeur de ValButton :

```
if ValButton == 1:
    Digital_Write(board,PinLedR,1)
else:
    Digital_Write(board,PinLedR,0)
```

- Fin du programme en appuyant sur Ctrl-C :

→ La DEL rouge est éteinte et le port série est fermé.

2.3 Gestion des sorties analogiques

Pour utiliser une broche de l'Arduino en sortie analogique (mode PWM), il faut au préalable la déclarer avec la commande suivante :

loop.run_until_complete(board.set_pin_mode_pwm(pin))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express** ",
- "**pin**" est le numéro de la broche du microcontrôleur que l'on souhaite déclarer en sortie analogique,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

Le plus simple est de créer une fonction qui sera appelée dans nos programmes en Python pour déclarer une broche en sortie analogique :

```
def Set_AnalogOutput_Pin(board,pin):
   loop.run_until_complete(board.set_pin_mode_pwm(pin))
```

Ainsi, l'instruction pour déclarer la broche "11", de l'objet "board", en sortie analogique est:

Set_AnalogOutput_Pin(board, 11)

La tension de la broche N°11 déclarée en sortie analogique est réglable en modifiant le rapport cyclique du signal PWM entre **val = 0** (0 %) et **val = 255** (100%) :

loop.run_until_complete(board.analog_write(pin, val))

→ si val = 0, alors la tension est de 0 V
→ si val = 255, alors la tension est de 5 V
→ si val = 128, alors la tension est de 2,5 V

De même, on peut définir une fonction pour l'écriture sur une sortie analogique :

```
def Analog_Write(board, pin, val):
   loop.run_until_complete(board.digital_write(pin, val))
```

Ainsi, l'instruction pour régler la sortie analogique "9", de l'objet "board", à 2,5 V est :

Analog_Write(board, 9, 128)

Exemple :

L'exemple d'application de la gestion des sorties analogiques avec "Firmata Express" est le même que celui avec "Firmata Standard".

Le programme ("LedRVBAnalogOutput.py" dans le dossier "Codes/PY/FirmataExpress") va demander à l'utilisateur de saisir un nombre entier entre 0 et 255 de façon à régler la tension de la broche sur laquelle la DEL rouge est connectée entre 0 et 5 V.

```
# Importations des librairies et définition de fonctions

from ConnectToArduino import *
import asyncio

def Set_AnalogOutput_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_pwm(pin))

def Analog_Write(board, pin, val):
    loop.run_until_complete(board.analog_write(pin, val))

def Arduino_Exit(board):
    loop.run_until_complete(board.shutdown())


# Déclaration des constantes et variables

PinLedR = 11


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

loop = asyncio.get_event_loop()
Set_AnalogOutput_Pin(board, PinLedR)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter\n")


# Boucle principale du programme

print("Veuillez saisir la luminosite de la DEL rouge (nombre entier entre 0 et 255):")

while True:
    try:
        ChoixRP=False
        while ChoixRP==False:
            saisie = input("")
            try:
                saisie = int(saisie)
                assert saisie >= 0 and saisie <= 255
                ChoixRP = True
            except AssertionError:
                print("Le nombre indiqué n'est pas entre 0 et 255 !")
                ChoixRP = False
            except:
                print("Vous n'avez pas saisi un nombre entre 0 et 255 !")

        Analog_Write(board,PinLedR,saisie)

    except KeyboardInterrupt:
        Analog_Write(board,PinLedR,0)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module ”**ConnectToArduino.py**”, contenant les fonctions de connexion à l’arduino via le protocole ”Firmata Express”,
. La bibliothèque **"asyncio"** nécessaire au fonctionnement de **"PymataExpress"**
. Fonction **" Set_AnalogOutput_Pin "** pour déclarer une broche en sortie analogique,
. Fonction **"Analog_Write"** pour modifier la tension d’une broche analogique,
. Fonction **"Arduino_Exit"** pour fermer le port COM et se déconnecter de l’Arduino

- Déclaration des constantes et variables :

. **PinLedR = 11** (constante correspondant au n° de la broche sur laquelle la DEL rouge est connectée)
. **ChoixRP =False** (variable booléenne indiquant si la saisie du rapport cyclique a été réalisée)
. **PortComArduino** (port COM sur lequel l’Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM : **PortComArduino = SelectPortCOM()**

. Tentative d’ouverture du port COM sélectionné et connexion à l’Arduino:

board = OpenPortCom(PortComArduino)

. si la connexion à l’Arduino est réussie:

→ Définition d’une boucle asyncio : **loop = asyncio.get_event_loop()**

→ Déclaration de la broche de la DEL rouge en sortie analogique :

Set_AnalogOutput_Pin(board, PinLedR)

- Boucle principale du programme (boucle "while True") :

. Demande de saisie de la valeur de la luminosité souhaitée entre 0 et 255 (un test est effectué sur la valeur saisie):

saisie = input("")

. La luminosité de la DEL rouge est réglée à la valeur saisie :

AnalogWrite(board,PinLedR,saisie)

- Fin du programme en appuyant sur Ctrl-C :

→ La DEL rouge est éteinte et le port série est fermé.

2.4 Gestion des entrées analogiques

Pour lire la valeur de la tension d'une entrée analogique (par exemple, la broche A0), il faut la déclarer au préalable en entrée analogique avec la commande suivante :

loop.run_until_complete(board.set_pin_mode_analog_input(0))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express**",
- "**0**" est le numéro de la broche A0 du microcontrôleur que l'on souhaite déclarer en entrée analogique,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

On peut définir une fonction déclarant plus facilement une broche en entrée analogique :

```
def Set_AnalogInput_Pin(board, pin):
  loop.run_until_complete(board.set_pin_mode_analog_input(pin))
```

La syntaxe pour déclarer la broche A0 en entrée numérique est alors plus simple :

Set_AnalogInput_Pin(board,0)

Ensuite, on pourra lire la valeur de la tension de la broche au moyen de cette instruction :

value = loop.run_until_complete(board.analog_read(0))

qui retourne une liste dont le premier élément (**value[0]**) est un nombre entier décimal entre 0 et 1023 représentatif d'une tension entre 0 et 5V.

De même, on peut définir une fonction pour lire la valeur de la tension d'une entrée analogique :

```
def Analog_Read(board, pin):
  value = loop.run_until_complete(board.analog_read(pin))
  return value[0]
```

Ainsi, l'instruction pour lire la valeur de la tension de la broche A0 devient :

ValA0 = Analog_Read(board, 0)

Exemple :

L'exemple d'application de la gestion des entrées analogiques avec "Firmata Express" est le même que celui avec "Firmata Standard".

Le programme ("LedRVBAnalogInput.py" dans le dossier "Codes/PY/FirmataExpress") suivant, lit la valeur de la broche A0 sur laquelle est connecté un potentiomètre et règle la luminosité de la DEL rouge proportionnellement à cette valeur :

```
# Importations des librairies et définition de fonctions

from ConnectToArduino import *
import asyncio

def Set_AnalogInput_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_analog_input(pin))

def Analog_Read(board, pin):
    value = loop.run_until_complete(board.analog_read(pin))
    return value[0]

def Set_AnalogOutput_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_pwm(pin))

def Analog_Write(board, pin, val):
    loop.run_until_complete(board.analog_write(pin, val))

def Arduino_Exit(board):
    loop.run_until_complete(board.shutdown())


# Déclaration des constantes et variables

PinLedR = 11
PinPot = 0
ValPot = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

loop = asyncio.get_event_loop()
Set_AnalogOutput_Pin(board, PinLedR)
Set_AnalogInput_Pin(board, PinPot)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter\n")


# Boucle principale du programme

while True:
    try:
        ValPot = Analog_Read(board,0)
        Analog_Write(board,PinLedR,int(ValPot/4))

    except KeyboardInterrupt:
        Analog_Write(board,PinLedR,0)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

  . Le module "**ConnectToArduino.py**", contenant les fonctions de connexion à l'arduino via le protocole "**Firmata Express**",
  . La bibliothèque **"asyncio"** nécessaire au fonctionnement de **"PymataExpress"**,
  . Fonction **"Set_AnalogOutput_Pin"** pour déclarer une broche en sortie analogique,
  . Fonction **"Set_AnalogInput_Pin"** pour déclarer une broche en entée analogique,
  . Fonction **"Analog_Write"** pour modifier la tension d'une sortie analogique,
  . Fonction **"Analog_Read"** pour lire la valeur de la tension d'une entrée analogique,
  . Fonction **"Arduino_Exit"** pour fermer le port COM et se déconnecter de l'Arduino.

- Déclaration des constantes et variables :

  . **PinLedR = 11** (constante correspondant au n° de la broche sur laquelle la DEL rouge est connectée)
  . **PinPot = 0** (constante correspondant au n° de la broche A0 sur laquelle le potentiomètre est connecté)
  . **ValPot = 0** (variable pour stocker la valeur de l'entrée analogique A0)
  . **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

  . Détection du port COM : **PortComArduino = SelectPortCOM()**

  . Tentative d'ouverture du port COM sélectionné et connexion à l'Arduino:

  board = OpenPortCom(PortComArduino)

  . si la connexion à l'Arduino est réussie:

  → Définition d'une boucle asyncio : **loop = asyncio.get_event_loop()**

  → Déclaration de la broche de la DEL rouge en sortie analogique :
  Set_AnalogOutput_Pin(board, PinLedR)

  → Déclaration de la broche du potentiomètre en entrée analogique :
  Set_AnalogInput_Pin(board, PinPot)

- Boucle principale du programme (boucle "while True") :

  . Lecture de la valeur (nombre décimal entre 0 et 1023) de la broche du potentiomètre :

  ValPot = Analog_Read(board,0)

  . La DEL rouge est connecté à une broche déclarée en sortie analogique qui accepte des valeurs entières entre 0 et 255. La valeur de la broche A0 lue par l'instruction précédente étant comprise entre 0 et 1023, celle-ci sera divisée par 4 et appliquée sur la broche de la DEL :

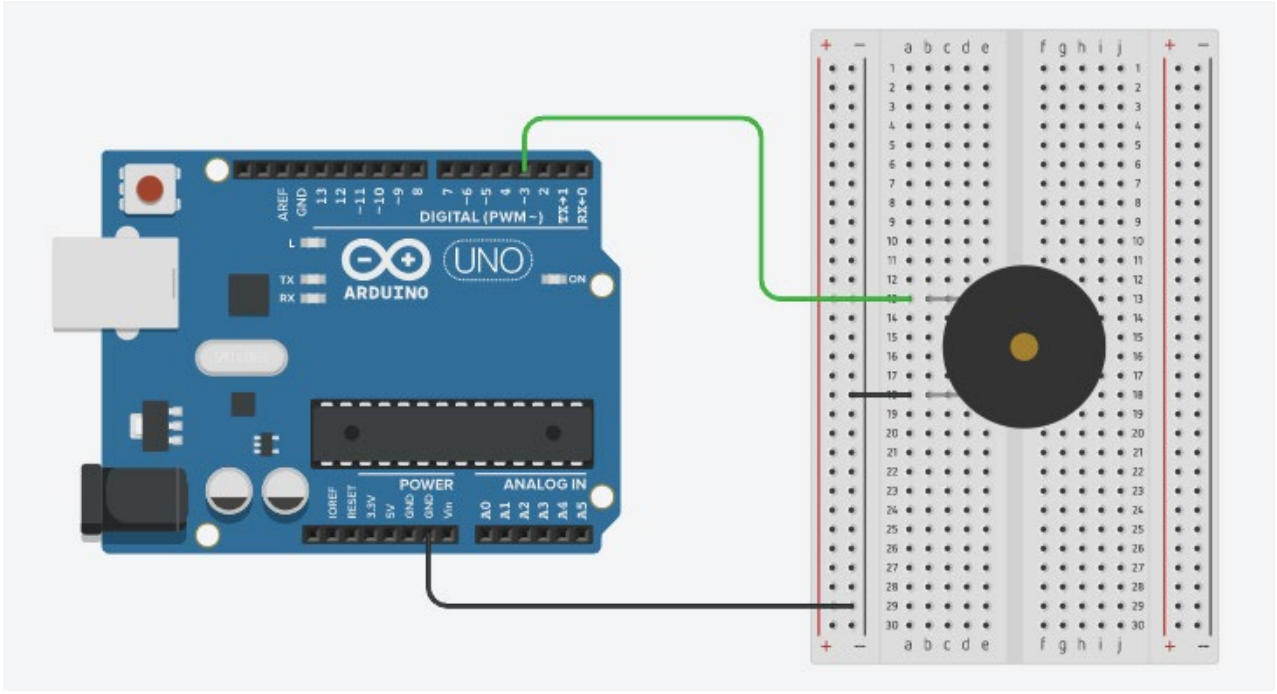
  AnalogWrite(board,PinLedR, int(ValPot/4))

- Fin du programme en appuyant sur Ctrl-C :

  → La DEL rouge est éteinte et le port série est fermé.

2.5 La prise en charge de la bibliothèque Arduino "Tone"

Pour produire un son avec un Arduino, on utilise un petit haut-parleur ou un buzzer (transducteur) piézo-électrique (communément appelé "piezo") connecté sur une des sorties de l'Arduino comme ci-dessous :



En langage Arduino, Le signal électrique appliqué par le microcontrôleur sur une de ses sorties digitales ou analogiques, sur laquelle est connecté le piezo ou le haut-parleur, est réalisé avec la fonction tone().

Cette fonction génère une onde carrée (onde symétrique avec "duty cycle" (niveau haut/période) à 50%) à la fréquence spécifiée en Hertz (Hz) sur une broche. La durée peut être précisée, sinon l'impulsion continue jusqu'à l'appel de l'instruction noTone().

Avec "**pymata-express**" pour utiliser une broche de l'Arduino en mode "tone", il faut au préalable la déclarer avec l'instruction :

loop.run_until_complete(board.set_pin_mode_tone(pin))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express**",
- "**pin**" est le numéro de la broche du microcontrôleur sur laquelle est connecté le piezo ou le haut-parleur,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

On peut définir une fonction déclarant plus facilement une broche en mode "tone" :

```
def Set_Tone_Pin(board,pin):
  loop.run_until_complete(board.set_pin_mode_tone(pin))
```

La syntaxe pour déclarer la broche N° 3 en mode "tone" est alors plus simple :

Set_Tone_Pin(board,3)

Ensuite, on peut produire un son de fréquence "**F**" en Hz pendant une durée "**D**" en ms avec un piezo connecté sur la broche "**pin**" au moyen de cette instruction :

loop.run_until_complete(board.play_tone(pin, F, D))

L'onde sonore peut être émise de façon continue :

loop.run_until_complete(board.play_tone_continuously(pin, F))

Le plus simple est de réunir les deux instructions dans une seule fonction dont les arguments sont le N° de la broche, la fréquence et la durée, puis d'utiliser l'une ou l'autre instruction en fonction de la valeur de la durée :

```
def Tone(board,pin,freq,duration):
  if duration>0:
    loop.run_until_complete(board.play_tone(pin, freq, duration))
  else:
    loop.run_until_complete(board.play_tone_continuously(pin, freq))
```

Ainsi, la commande pour émettre un son, avec un piezo connecté à la broche N°3, à une fréquence de 440 Hz est :

- Pendant 1 s : **Tone(board,3,440,1000)**
- De façon continue : **Tone(board,3,440,0)**

Une onde sonore émise en continu est arrêtée avec l'instruction :

loop.run_until_complete(board.play_tone_off(pin))

où "**pin**" est le numéro de la broche du microcontrôleur sur laquelle est connecté le piezo ou le haut-parleur.

On définira une fonction utilisant cette instruction :

```
def No_Tone(board,pin):
   loop.run_until_complete(board.play_tone_off(pin))
```

Et pour arrêter l'émission sonore sur la broche N°3, l'instruction devient :

No_Tone(board,3)

Exemple :

Le programme d'application ("Tone.py" dans le dossier "Codes/PY/FirmataExpress") suivant, émet un beep sonore :

```
# Importations des librairies et définition de fonctions

from ConnectToArduino import *
import asyncio
import time

def Tone(board,pin,freq,duration):
    if duration>0:
        loop.run_until_complete(board.play_tone(pin, freq, duration))
    else:
        loop.run_until_complete(board.play_tone_continuously(pin, freq))

def No_Tone(board,pin):
    loop.run_until_complete(board.play_tone_off(pin))

def Set_Tone_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_tone(pin))

def Arduino_Exit(board):
    loop.run_until_complete(board.shutdown())


# Déclaration des constantes et variables

PinTone = 3
FreqTone = 440
TimeSleep1 = 0.5
TimeSleep2 = 0.5


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)

# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

loop = asyncio.get_event_loop()
Set_Tone_Pin(board, PinTone)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter\n")
```

```
# Boucle principale du programme

while True:
    try:
        Tone(board, PinTone, FreqTone, 0)
        time.sleep(TimeSleep1)
        No_Tone(board, PinTone)
        time.sleep(TimeSleep2)

    except KeyboardInterrupt:
        No_Tone(board, PinTone)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module ”**ConnectToArduino.py**”, contenant les fonctions de connexion à l’arduino via le protocole ”**Firmata Express**”,

. La bibliothèque **"asyncio"** nécessaire au fonctionnement de **"PymataExpress"**,

. La bibliothèque "**time**" pour la gestion des durées d’émission et de silence.

. Fonction **"Set_Tone_Pin"** pour déclarer une broche en mode **”Tone”**,

. Fonction **"Tone"** pour produire une onde sonore de fréquence fixée en Hz,

. Fonction **"No_Tone"** pour arrêter l’émisssion sonore,

. Fonction **"Arduino_Exit"** pour fermer le port COM et se déconnecter de l’Arduino.

- Déclaration des constantes et variables :

. **PinTone = 3** (constante correspondant au n° de la broche sur laquelle le piezo est connecté)

. **FreqTone = 440** (variable correspondant à la fréquence en Hz de l’onde sonore)

. **TimeSleep1 = 0.5** (variable correspondant à la durée d’émission du son en s)

. **TimeSleep2 = 0.5** (variable correspondant à la durée du silence en s)

. **PortComArduino** (port COM sur lequel l’Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM : **PortComArduino = SelectPortCOM()**

. Tentative d’ouverture du port COM sélectionné et connexion à l’Arduino:

board = OpenPortCom(PortComArduino)

. si la connexion à l’Arduino est réussie:

→ Définition d’une boucle asyncio : **loop = asyncio.get_event_loop()**

→ Déclaration de la broche du piezo en mode **"Tone"** :
Set_Tone_Pin(board, PinTone)

- Boucle principale du programme (boucle "while True") :

. Si la connexion à l’Arduino est réussie, émission de l’onde sonore à la fréquence "FreqTone" pendant "TimeSleep1", puis silence pendant "TimeSleep2", puis nouvelle émission, etc…

- Fin du programme en appuyant sur Ctrl-C :

→ Arrêt de l’émission sonore et le port série est fermé.

2.6 La prise en charge des capteurs ultrasoniques

Les capteurs ultrasoniques sont composés d'un émetteur d'ultrasons, d'un récepteur d'ultrasons et d'un circuit de commande. Ils sont généralement utilisés pour mesurer une distance entre le capteur et un obstacle.

Le capteur ultrasonique, le plus souvent utilisé avec un Arduino, le **HC-SR04**, dispose de 2 broches différentes pour l'émission et la réception des ultrasons.

Il existe également des capteurs ultrasoniques à une seule broche, comme le Grove **101020010**.

Avec "**pymata-express 1.4**", pour utiliser un capteur ultrasonique, il faut déclarer les broches servant à l'émission et à la réception des ultrasons avec la commande :

loop.run_until_complete(board.set_pin_mode_sonar(trigger_pin, echo_pin, Get_Distance))

où :

- "**board**" est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express**",
- "**trigger_pin**" est le numéro de la broche du microcontrôleur sur laquelle est connecté l'émetteur d'ultrasons,
- "**echo_pin**" est le numéro de la broche du microcontrôleur sur laquelle est connecté le récepteur d'ultrasons,
- **loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

  loop = asyncio.get_event_loop(),

- "**Get_Distance**" est une fonction "**asyncio**" pour récupérer la valeur de la distance en cm entre le capteur et un obstacle :

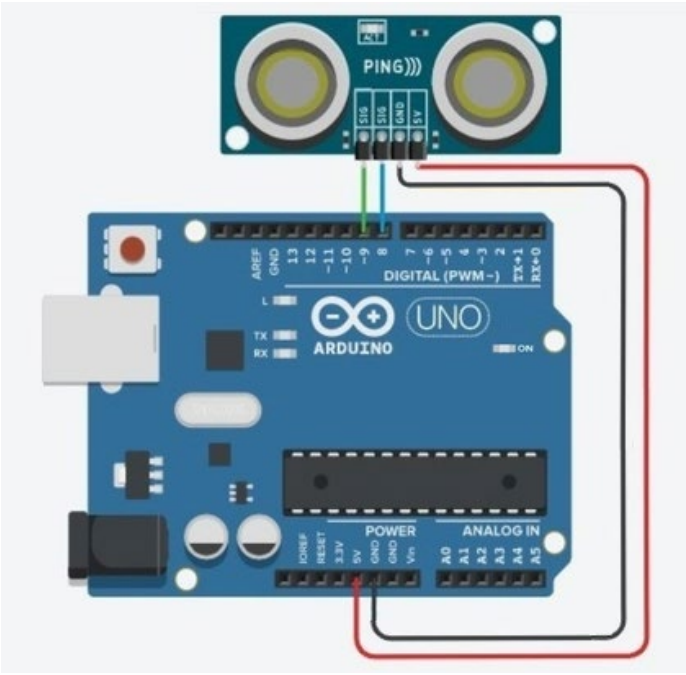
```
async def Get_Distance(data):
    global Distance
    Distance = data[1]
```

Remarques :

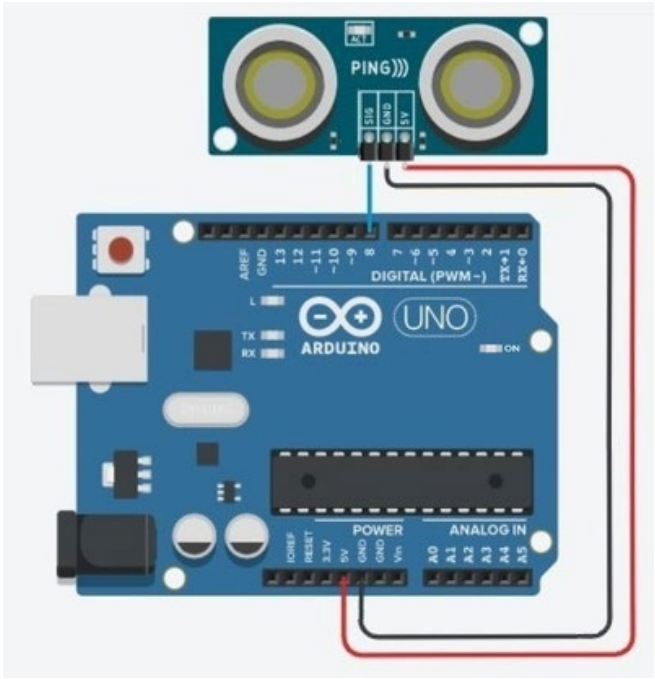
. "**data**" est une liste de 2 éléments (nombres décimaux entiers) générée lors de la déclaration des broches du capteur dont le premier élément (data[0]) est la valeur de "**trigger_pin**" et le deuxième élément (data[1]) est la distance mesurée en cm.

. Dans le cas d'un capteur ultrasonique à une broche commune pour l'émission et la réception des ultrasons, il suffit de déclarer la même broche pour "**trigger_pin**" et "**echo_pin**".

Un capteur ultrasonique à 2 broches différentes pour l'émission et la réception des ultrasons est connecté à l'Arduino comme ci-dessous :



Un capteur ultrasonique à 1 broche commune pour l'émission et la réception des ultrasons est connecté à l'Arduino comme ci-dessous :



On peut définir une fonction déclarant plus facilement les broches d'un capteur ultrasonique :

```
def Set_Sonar_Pins(board, trigger_pin, echo_pin):
    loop.run_until_complete(board.set_pin_mode_sonar(trigger_pin, echo_pin, Get_Distance))
```

Ainsi, dans le cas d'un capteur ultrasonique à 2 broches différentes pour l'émission (broche 8) et la réception (broche 9) des ultrasons, l'instruction de déclaration est :

Set_Sonar_Pins(board, 8, 9)

Pour obtenir la valeur de la distance en cm entre le capteur et l'obstacle, on utilise la fonction "**sonar_read**" qui fait appel à la fonction "**Get_Distance**" définie lors de la déclaration des broches du capteur :

loop.run_until_complete(board.sonar_read(trigger_pin))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express**",
- "**trigger_pin**" est le numéro de la broche du microcontrôleur sur laquelle est connecté l'émetteur ultrasons.

Si on définit une fonction, comme ci-dessous :

```
def Sonar_Read(board, trigger_pin):
   loop.run_until_complete(board.sonar_read(trigger_pin))
```

Pour obtenir la valeur de la distance en cm entre le capteur (connecté aux broches 8 et 9) et l'obstacle, on appelle la fonction :

Sonar_Read(board,8)

La variable globale "**Distance**" est alors mise à jour via la fonction "**Get_Distance**", celle-ci étant appelée lors de l'appel de la fonction " **Sonar_Read** ".

Exemple :

Le programme d'application ("Sonar.py" dans le dossier "Codes/PY/FirmataExpress") suivant, mesure la distance entre un capteur ultrasonique à 2 broches (comme le **HC-SR04)** et un obstacle, et l'affiche dans la console Python :

```
# Importations des librairies et définition de fonctions

from ConnectToArduino import *
import asyncio
import time

async def Get_Distance(data):
    global Distance
    Distance = data[1]

def Set_Sonar_Pins(board, trigger_pin, echo_pin):
    loop.run_until_complete(board.set_pin_mode_sonar(trigger_pin, echo_pin, Get_Distance))

def Sonar_Read(board, trigger_pin):
    loop.run_until_complete(board.sonar_read(trigger_pin))

def Arduino_Exit(board):
    loop.run_until_complete(board.shutdown())


# Déclaration des constantes et variables

TRIGGER_PIN = 8
ECHO_PIN = 9
Distance = 0
OldDistance = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

loop = asyncio.get_event_loop()
Set_Sonar_Pins(board, TRIGGER_PIN, ECHO_PIN)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter\n")


# Boucle principale du programme

while True:
    try:
        Sonar_Read(board,TRIGGER_PIN)
        if Distance != OldDistance:
            print("Distance (cm):",Distance)
            OldDistance = Distance
        time.sleep(.01)

    except KeyboardInterrupt:
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

  . Le module "**ConnectToArduino.py**", contenant les fonctions de connexion à l'arduino via le protocole "**Firmata Express**",
  . La bibliothèque **"asyncio"** nécessaire au fonctionnement de **"PymataExpress"**,
  . La bibliothèque "**time**" pour la gestion des durées d'émission et de silence.
  . Fonction **"Get_Distance"** pour récupérer la valeur de la distance entre le capteur et l'obstacle,
  . Fonction **"Set_Sonar_Pins"** pour déclarer les broches du capteur,
  . Fonction **"Sonar_Read"** pour effectuer la mesure de la distance,
  . Fonction **"Arduino_Exit"** pour fermer le port COM et se déconnecter de l'Arduino.

- Déclaration des constantes et variables :

  . **TRIGGER_PIN = 8** (constante correspondant au n° de la broche sur laquelle l'émetteur est connecté)
  . **ECHO_PIN = 9** (constante correspondant au n° de la broche sur laquelle le récepteur est connecté)
  . **Distance = 0** (variable correspondant à la distance entre le capteur et l'obstacle)
  . **OldDistance = 0** (variable correspondant à la valeur précédente de la distance)
  . **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

  . Détection du port COM : **PortComArduino = SelectPortCOM()**

  . Tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

  board = OpenPortCom(PortComArduino)

  . Si la connexion à l'Arduino est réussie :

  → Définition d'une boucle asyncio : **loop = asyncio.get_event_loop()**

  → Déclaration des broches du capteur ultrasonique :
  Set_Sonar_Pins(board, TRIGGER_PIN, ECHO_PIN)

- Boucle principale du programme (boucle "while True") :

  . Mesure de la distance : **Sonar_Read(board,TRIGGER_PIN)**

  → Affichage de la distance dans la console Python si la valeur mesurée est différente de la valeur précédente.

- Fin du programme en appuyant sur Ctrl-C :

  → Le port série est fermé.

- Résultats dans la fenêtre Python Shell :

```
Pymata Express Version 1.4
Copyright (c) 2018-2019 Alan Yorinks All rights reserved.

Opening COM3 ...
Waiting 4 seconds for the Arduino To Reset.

Arduino found and connected to COM3

Retrieving Arduino Firmware ID...
Arduino Firmware ID: 2.5 FirmataExpress.ino
Auto-discovery complete. Found 20 Digital Pins and 6 Analog Pins


Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter

Distance (cm): 27
Distance (cm): 26
Distance (cm): 27
Distance (cm): 26
Distance (cm): 25
Distance (cm): 24
Distance (cm): 23
Distance (cm): 22
Distance (cm): 21
Distance (cm): 20
Distance (cm): 21
Distance (cm): 20
```