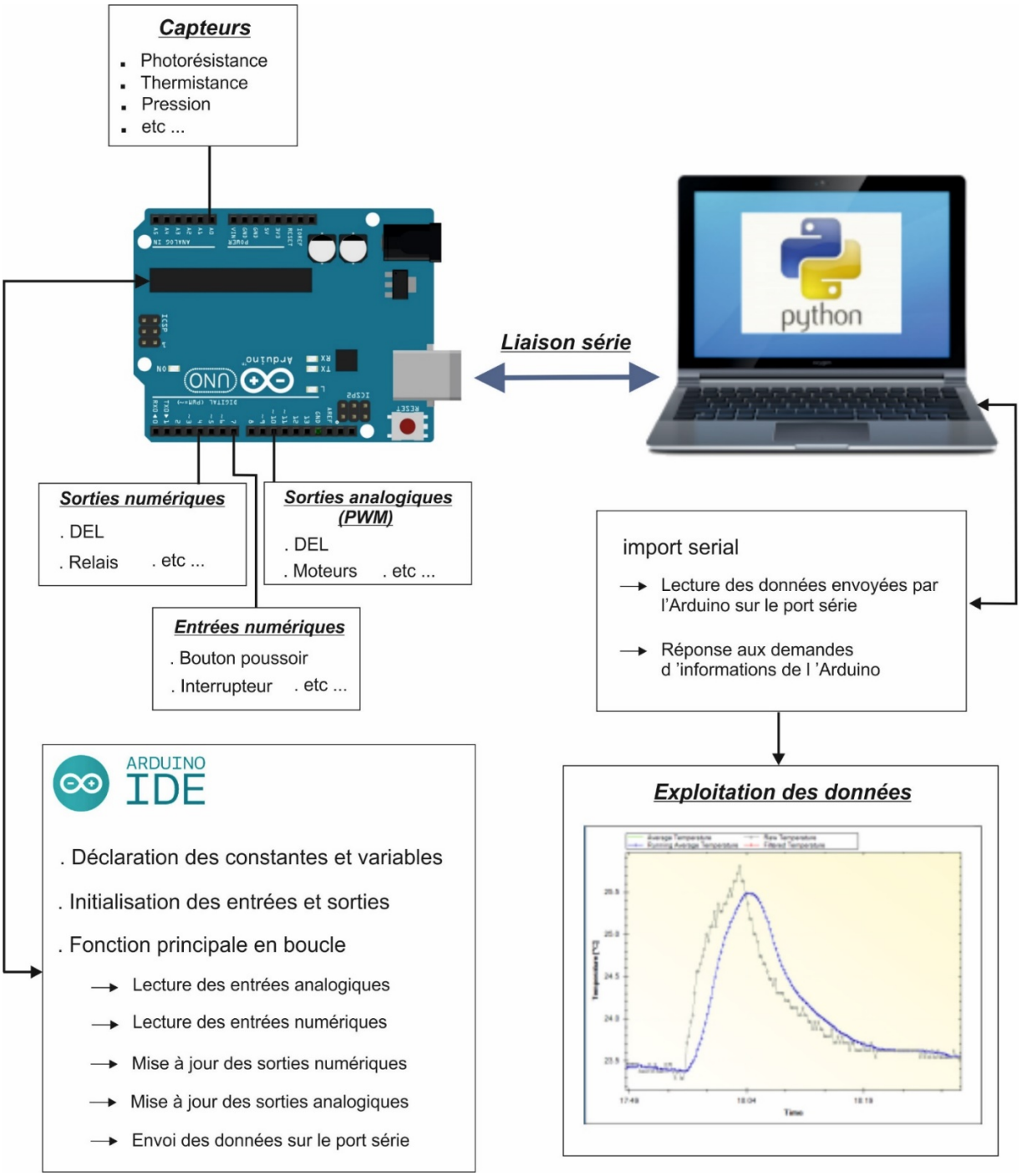
Synthèse - Conclusion

Les méthodes de communications entre un arduino et un programme Python sont résumées à l'aide des schémas suivant :

- Communication Arduino - Python via le port série avec Pyserial :

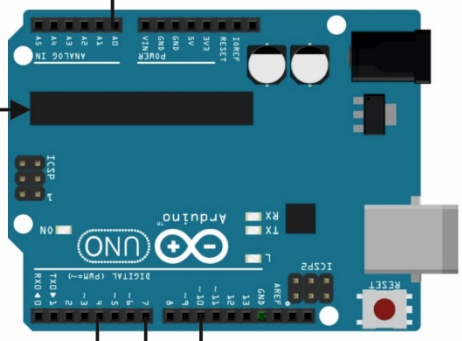


- L'Arduino envoie des données sur le port série par la fonction **"Serial.print()"**
- Le programme Python réceptionne les données de la liaison série, envoyées par l'Arduino, à l'aide de la fonction **"readline()"**
- Le programme Python envoie des données sur le port série avec la fonction **"write()"**
- L'arduino réceptionne les données envoyées par le programme Python sur la liaison série grâce à la fonction **"Serial.read()"**

- Communication Arduino-Python via le protocole de communication "Firmata"

Capteurs

- Photorésistance
- Thermistance
- Pression
- etc ...



Liaison série

Sorties numériques

. DEL

. Relais . etc ...

Sorties analogiques (PWM)

. DEL

. Moteurs . etc ...

Entrées numériques

. Bouton poussoir

. Interrupteur . etc ...

ARDUINO IDE

Firmata standard

ou Firmata Express

import pyfirmata

ou import PymataExpress

. Déclaration des constantes et variables

. Initialisation des entrées et sorties

. Fonction principale en boucle

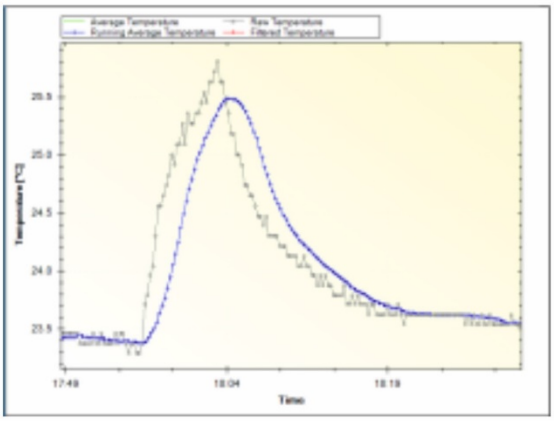
→ Lecture des entrées analogiques

→ Lecture des entrées numériques

→ Mise à jour des sorties numériques

→ Mise à jour des sorties analogiques

Exploitation des données



. Firmata standard

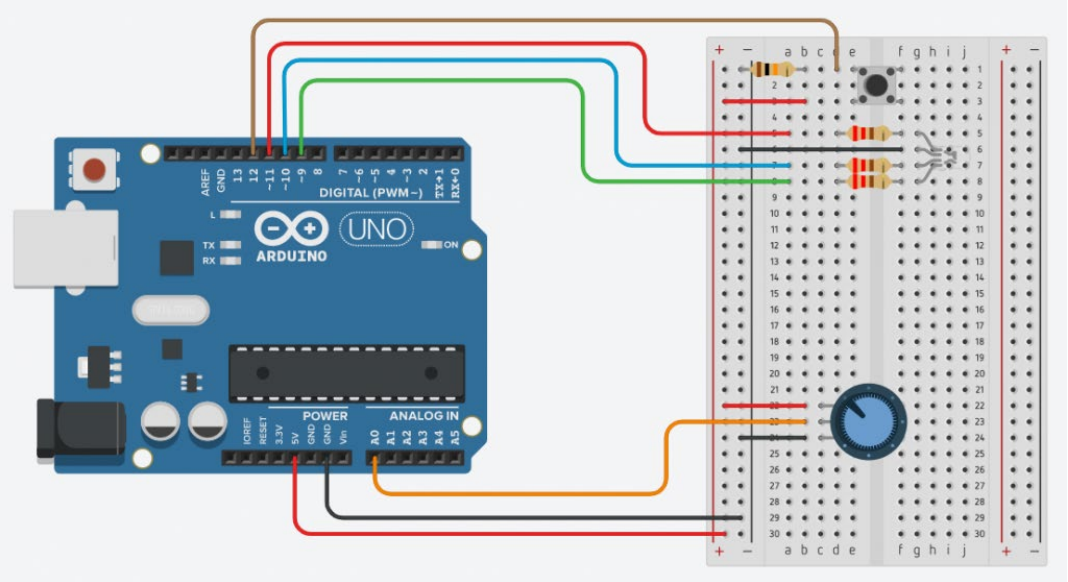
Pour contrôler l'Arduino via le protocole de communication Firmata standard, le programme Python utilise la bibliothèque **"PyFirmata"** (le sketch **"Firmata standard"** doit être téléversé dans la mémoire de l'arduino au préalable).

Toutes les fonctions utiles à l'utilisation de **"PyFirmata"** (fonction de déclaration des entrées et sorties, d'itérateur, d'écritures…) que nous avons définies jusqu'à présent ont été regroupées dans un fichier Python, nommé **"PyFirmataDef.Py"** que l'on peut importer dans tous les programmes, à condition que le fichier des fonctions soit dans le même dossier que le fichier du programme, avec l'instruction :

from PyFirmataDef import *

```
import pyfirmata


########## GESTION DES SORTIES NUMERIQUES ##########

# MODIFICATION DE L'ETAT LOGIQUE D'UNE SORTIE NUMERIQUE

def DigitalWrite(board,pin,val):
    board.digital[pin].write(val)



########## GESTION DES ENTREES NUMERIQUES ##########

# DECLARATION D'UNE BROCHE EN ENTREE NUMERIQUE
# LECTURE DE L'ETAT LOGIQUE AVEC LA FONCTION DigitalInputPin.read()

def DigitalInput(board,pin):
    DigitalInputPin=board.get_pin('d:'+ str(pin) +':i')
    return DigitalInputPin



########## GESTION DES SORTIES ANALOGIQUES ##########

# DECLARATION D'UNE BROCHE EN SORTIE ANALOGIQUE

def AnalogOutput(board,pin):
    AnalogOutputPin=board.get_pin('d:'+ str(pin) +':p')
    return AnalogOutputPin

# MODIFICATION DE LA VALEUR D'UNE SORTIE ANALOGIQUE

def AnalogWrite(board,pin,val):
    board.digital[pin].write(val)



########## GESTION DES ENTREES ANALOGIQUES ##########

# DECLARATION D'UNE BROCHE EN ENTREE ANALOGIQUE
# LECTURE DE LA VALEUR DE L'ENTREE ANALOGIQUE AVEC LA FONCTION AnalogInputPin.read()

def AnalogInput(board,pin):
    AnalogInputPin = board.get_pin('a:'+ str(pin) +':i')
    return AnalogInputPin



########## ITERATEUR ##########

# DECLARATION D'UN ITERATEUR POUR EVITER LA SATURATION DU PORT SERIE

def Iterateur(board):
    it = pyfirmata.util.Iterator(board)
    it.start()
    return it
```

Le programme (SyntheseFirmataStandard.py) ci-dessous met en application toutes ces fonctions avec le circuit suivant :



Ce circuit est composé :

. d'une DEL RVB dont les DELs rouge, verte et bleue sont respectivement connectés aux sorties, 11, 9 et 10 de l'Arduino,

. d'un potentiomètre, dont le « point milieu » est relié à l'entrée analogique A0 du microcontrôleur,

. et d'un bouton poussoir dont la sortie est reliée à la borne 12 de l'Arduino.

Avec ce programme:

. La DEL rouge est allumée ou éteinte avec le bouton poussoir,

. La luminosité de la DEL bleue est réglée avec le potentiomètre.

```
# Importations des librairies et définition de fonctions

from PyFirmataDef import *
from ConnectToArduino import *
import time


# Déclaration des constantes et variables

PinLedR = 11
PinLedB = 10
PinButton = 12
ValButton = 0
PinPot = 0
ValPot = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


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

DigitalInputPin = DigitalInput(board, PinButton)
AnalogInputPin = AnalogInput(board, PinPot)
PinPWM = AnalogOutput(board, PinLedR)

ArduinoIterateur = Iterateur(board)
time.sleep(0.5)

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


# Boucle principale du programme

while True:
    try:
        ValButton = DigitalInputPin.read()
        if ValButton == 1:
            DigitalWrite(board,PinLedB,1)
        else:
            DigitalWrite(board,PinLedB,0)
        ValPot = AnalogInputPin.read()
        AnalogWrite(board,PinLedR,ValPot)

    except KeyboardInterrupt:
        DigitalWrite(board,PinLedB,0)
        AnalogWrite(board,PinLedR,0)
        board.exit()
        sys.exit(0)
```

. Firmata Express

Pour contrôler l'Arduino via le protocole de communication Firmata Express, le programme Python utilise la bibliothèque **"pymata-express"** (le sketch **"Firmata Express"** doit être téléversé dans la mémoire de l'Arduino au préalable).

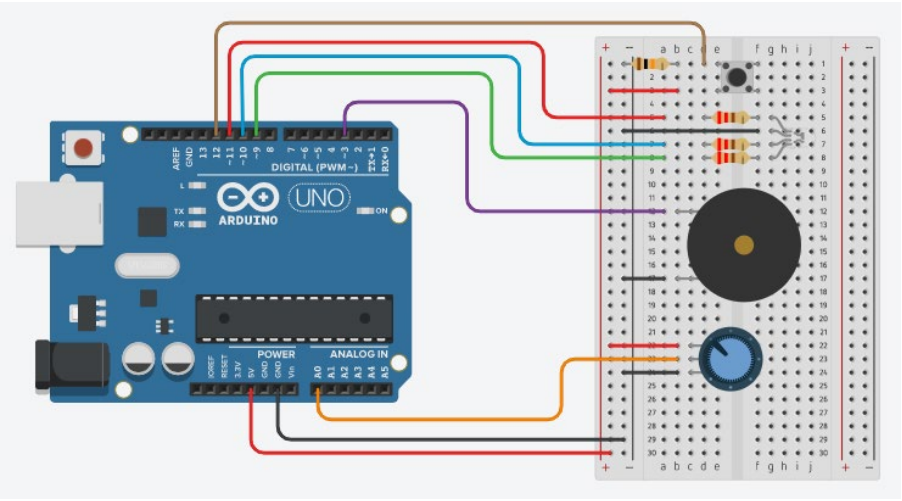
Toutes les fonctions utiles à l'utilisation de **"Pymata-express"** (fonction de déclaration des entrées et sorties, de lectures, d'écritures…) que nous avons définies jusqu'à présent ont été regroupées dans un fichier Python, nommé **"PymataExpressDef.Py"** que l'on peut importer dans tous les programmes, à condition que le fichier des fonctions soit dans le même dossier que le fichier du programme, avec l'instruction :

from PymataExpressDef import *

```
import asyncio


########## DEFINITION D'UNE BOUCLE ASYNCIO ##########

loop = asyncio.get_event_loop()


########## GESTION DES ENTREES NUMERIQUES ##########

# DECLARATION D'UNE BROCHE EN ENTREE NUMERIQUE

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

# LECTURE DE L'ETAT LOGIQUE D'UNE BROCHE DECLAREE EN ENTREE NUMERIQUE

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


########## GESTION DES SORTIES NUMERIQUES ##########

# DECLARATION D'UNE BROCHE EN SORTIE NUMERIQUE

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

# MODIFICATION DE L'ETAT LOGIQUE D'UNE SORTIE NUMERIQUE

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


########## GESTION DES ENTREES ANALOGIQUES ##########

# DECLARATION D'UNE BROCHE EN ENTREE ANALOGIQUE

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

# LECTURE DE LA VALEUR DE L'ENTREE ANALOGIQUE

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

```
########## GESTION DES SORTIES ANALOGIQUES ##########

# DECLARATION D'UNE BROCHE EN SORTIE ANALOGIQUE

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

# MODIFICATION DE LA VALEUR D'UNE SORTIE ANALOGIQUE

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


########## GESTION DU SON ##########

# DECLARATION D'UNE BROCHE EN MODE TONE

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

# EMISSION SONORE

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))

# ARRET DE L'EMISSION SONORE

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


########## DECONNEXION DE L'ARDUINO ##########

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

Avec le même circuit que celui du programme de synthèse du protocole Firmata Standard, auquel, un buzzer a été ajouté sur la broche N°3 de l'Arduino, le programme, nommé "**SynthèseFirmataExpress.py**" met en application toutes ces fonctions :



```
# Importations des librairies et définition de fonctions

from PymataExpressDef import *
from ConnectToArduino import *


# Déclaration des constantes et variables

PinLedR = 11
PinLedB = 10
PinTone = 3
FreqTone = 440
PinPot = 0
ValPot = 0
PinButton = 12
ValButton = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


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

Set_AnalogOutput_Pin(board, PinLedR)
Set_AnalogInput_Pin(board, PinPot)
Set_DigitalOutput_Pin(board, PinLedB)
Set_DigitalInput_Pin(board, PinButton)
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:
        ValButton = Digital_Read(board, PinButton)
        if ValButton == 1:
            Digital_Write(board,PinLedB,1)
            Tone(board, PinTone, FreqTone, 0)
        else:
            Digital_Write(board,PinLedB,0)
            No_Tone(board, PinTone)

        ValPot = Analog_Read(board,0)
        Analog_Write(board,PinLedR,int(ValPot/4))

    except KeyboardInterrupt:
        Analog_Write(board,PinLedR,0)
        Digital_Write(board,PinLedB,0)
        No_Tone(board, PinTone)
        Arduino_Exit(board)
        sys.exit(0)
```

Avec ce programme:

. La DEL rouge est allumée ou éteinte avec le bouton poussoir,

. La luminosité de la DEL bleue est réglée avec le potentiomètre,

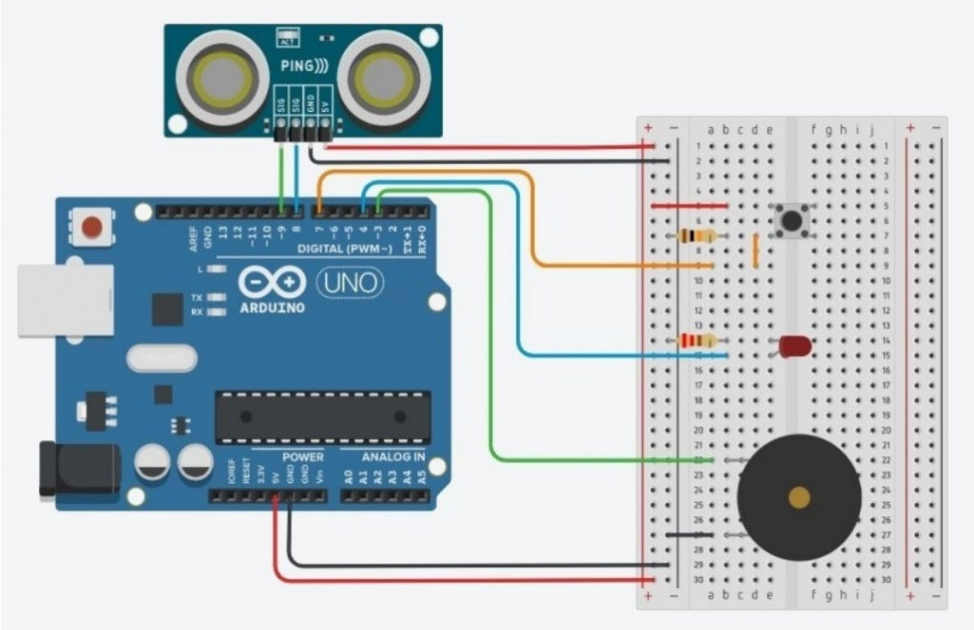
. Un son est émis en appuyant sur le bouton poussoir.

<u>Remarque</u> :

Comme vous avez pu le constater, le fichier Python, nommé **”PymataExpressDef.Py”**, regroupant les fonctions utiles à l’utilisation de **”Pymata-express”**, ne contient pas les fonctions gérant les capteurs ultrasoniques.

En effet, ces fonctions font intervenir une fonction **asyncio** (”**Get_Distance**”) qui utilise une variable globale et qui doit donc se trouver dans le fichier du programme principal.

Il est cependant possible d’importer les autres fonctions du fichier **”PymataExpressDef.Py”** afin de pouvoir les utiliser conjointement avec les fonctions de gestion des capteurs ultrasoniques, comme dans l’exemple ci-dessous (programme **”Sonar2.py”**) avec ce circuit :



Le programme permet de mesurer la distance entre un capteur ultrasonique à 2 broches (comme le HC-SR04) et un obstacle, et de l’afficher dans la console Python, quand le bouton poussoir est appuyé :

```
# Importations des librairies et définition de fonctions

from ConnectToArduino import *
from PymataExpressDef import *
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))


# Déclaration des constantes et variables

TRIGGER_PIN = 8
ECHO_PIN = 9
Distance = 0
OldDistance = 0
PinButton = 7
ValButton = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


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

Set_Sonar_Pins(board, TRIGGER_PIN, ECHO_PIN)
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 pousssoir pour mesurer une distance.\n")

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        if ValButton == 1:
            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)
```

Conclusion :

Le principal avantage de la méthode de communication par "**Firmata**" est que toute la gestion de l'Arduino se fait au sein du programme en Python.

En effet, une fois le programme "pilote" (Firmata standard ou Express) téléversé dans la mémoire de l'Arduino, il n'est plus nécessaire de le changer, puisque c'est le programme en Python qui sera modifié en fonction du matériel connecté au microcontrôleur.

Cependant, dans le cas d'utilisation de certains capteurs (par exemple, le capteur de température et d'humidité DHT 11), avec des bibliothèques uniquement développées pour le langage "Arduino IDE", il est plus simple d'utiliser un programme spécifique téléversé dans la mémoire de l'Arduino et de communiquer avec lui à l'aide de "**pyserial**".

Et si l'Arduino doit être utilisé de façon autonome (sans liaison série avec un ordinateur), il est possible d'utiliser un module (un shield) ajoutant une carte SD à l'Arduino sur laquelle des données pourront être enregistrées et exploitées ultérieurement.

Finalement, quel que soit la méthode de communication utilisée, le programme en python d'analyse et d'exploitation des données reste le même.

L'essentiel concernant la communication entre un Arduino et un programme Python ayant été vu, il est maintenant possible de développer des projets regroupant quelques activités programmées en Python pour les Arduino Uno.