Communication Arduino - Python via le port série

Python dispose d'une bibliothèque appelée, **pyserial**, qui permet d'accéder au port série. Un programme en python, **en remplacement du moniteur série**, pourra alors lui aussi lire ou écrire des données via le port série en utilisant cette bibliothèque.

La bibliothèque **"pyserial"** peut être installée via **"pip"**, à l'aide de la ligne de commande :

pip install pyserial

Pour utiliser la bibliothèque **"pyserial"** dans un programme python, il faut :

- importer le module **"serial**", à l'aide de l'instruction : **import serial**

- créer un objet **"port série**" et ouvrir le port avec la fonction **"Serial"** en précisant le port COM sur lequel l'Arduino est connecté, la vitesse de transmission des données (par exemple, 9600 baups) et éventuellement le temps d'attente en s pour la réception des données (par exemple, timeout=1) :

SerialPort = serial.Serial("COM 5", baudrate=9600, timeout=1)

1. Réception de données envoyées depuis l'Arduino par un programme Python

La réception des données envoyées depuis l'Arduino est effectuée à l'aide la fonction **"readline()" :**

Donnees = str(SerialPort.readline())

Cette instruction permet de lire les données du tampon série, de les convertir en chaine de caractères, à l'aide de la fonction **"str()"** et de stocker la chaine dans la variable **"Donnees"**.

La fonction **"readline()"** ne fonctionne que si un délai d'attente pour la réception des données a été déclaré (timeout de l'objet **"port série**").

1.1 Programme pour la réception d'une chaine de caractères

Reprenons le programme (nommé "AnalogRead.ino" dans le dossier "Codes/INO") à téléverser dans la mémoire de l'Arduino qui affiche, dans le moniteur série, la valeur de l'entrée analogique **A0** avec le circuit support des exemples.

En remplacement du moniteur série, voici le programme python (nommé "ReadLine.py" dans le dossier "Codes/PY") permettant de lire les données envoyées sur le port série et de les afficher dans la console Python :

```
# Programme associé avec "AnalogRead.ino"

# Importation des modules

from ConnectToArduino import *

# Déclaration des constantes et variables

donnees = ''

# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
SerialPort = OpenPortCom(PortComArduino)

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


# Boucle principale du programme

while True:

    try:
        donnees=str(SerialPort.readline())
        print(donnees)

    except KeyboardInterrupt:
        SerialPort.close()
        sys.exit(0)
```

<u>Déroulement du programme</u> :

- Importation des modules :

  Le module **"ConnectToArduino.py",** contenant les fonctions de connexion à l'arduino via le port série, est importé.

  L'importation de ce module importe également les bibliothèques :

  . serial (pour la connexion au port série de l'Arduino)

  . serial.tools.list_ports (pour déteminer les ports coms disponibles)

  . sys (pour mettre fin au programme en cas de problème de connexion au port série)

- Déclaration des constantes et variables :

  . **donnees** = '' (variable pour stocker les données qui transitent par le port série)

- Connexion à l'Arduino :

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

  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 et le programme est arrêté,

→ 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 avec la fonction "**OpenPortCom**" du module "**ConnectToArduino.py**":

SerialPort = OpenPortCom(PortComArduino)

→ message d'erreur et le programme est arrêté si l'ouverture du port com sélectionné a échoué.

```
def OpenPortCom(PortCom):

    try:
        PortSerial = serial.Serial(PortCom, baudrate=9600, timeout=1)
        return PortSerial

    except :
        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
        saisie = ""
        while saisie != "q":
            saisie = str(input("Entrez 'q' pour quitter: "))
        sys.exit()
```

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

. Lecture des données du tampon de la liaison série si la connexion à l'Arduino est réussie

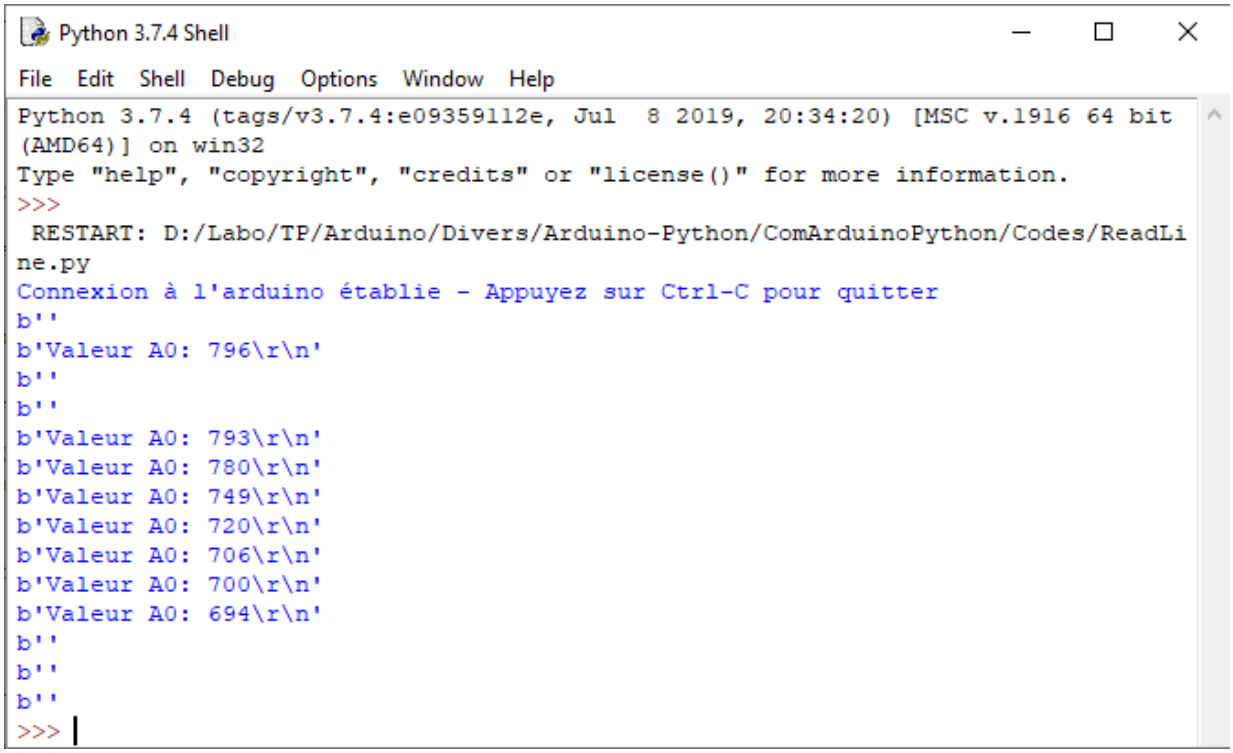
donnees=str(SerialPort.readline())

. Affichage des données dans la console Python :

print(donnees)

- Fin du programme en appuyant sur **Ctrl-C**

Et le résultat dans la console python :



Remarques :

- Dans la console Python, les données converties en chaine de caractères sont affichées entre des balises : **b' '** et le caractère "retour à la ligne" (**"\r\n"**) est également affiché.

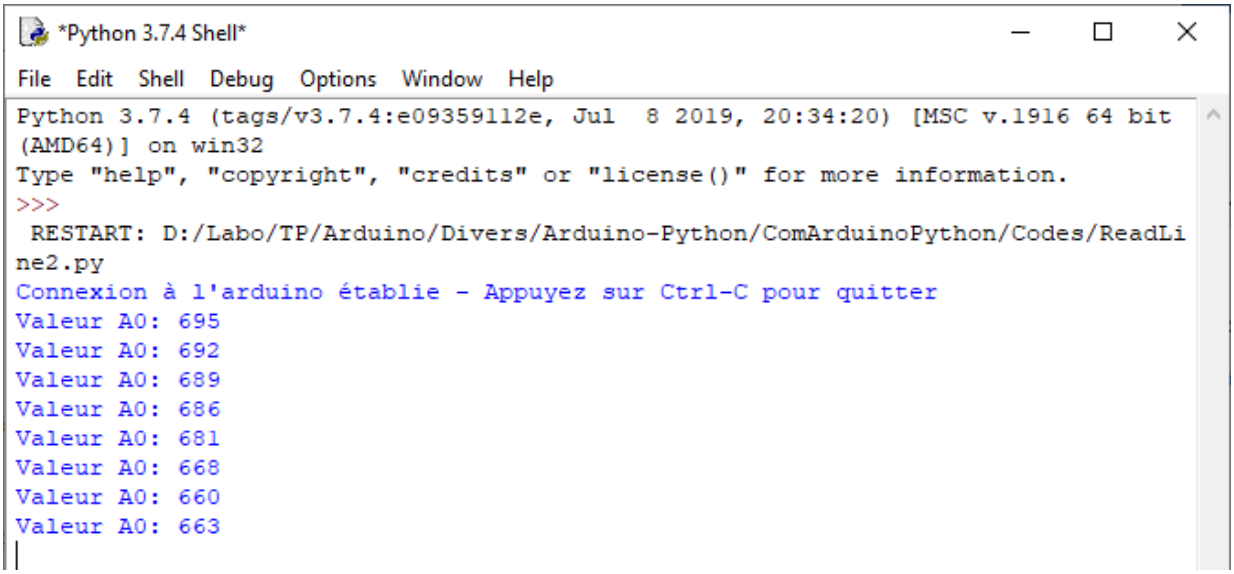
- Quand le délai d'attente de réception des données (**"timeout"**) est écoulé, une balise vide (**b' '**) est affichée.

- Pour supprimer les balises vides et n'afficher que les données envoyées par l'Arduino, il suffit de faire un traitement sur la chaine de caractères :

→ modification du programme **"ReadLine.py"** (programme **"ReadLine2.py"** dans le dossier "Codes/PY") :

```
# Programme associé avec "AnalogRead.ino"

# Importation des modules

from ConnectToArduino import *

# Déclaration des constantes et variables

donnees = ''

# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
SerialPort = OpenPortCom(PortComArduino)

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


# Boucle principale du programme

while True:

    try:
        donnees=str(SerialPort.readline())
        if donnees!="b''":
            donnees=donnees[2:len(donnees)-5]
            print(donnees)

    except KeyboardInterrupt:
        SerialPort.close()
        sys.exit(0)
```

Ainsi les 2 premiers caractères et les 5 derniers ne sont pas affichés :



1.2 Programme pour la conversion d'une chaine de caractères en nombre

Pour pouvoir exploiter, par un programme Python, les données envoyées par l'Arduino, dans la plupart des cas, il va falloir convertir la chaine de caractères reçues en nombre entier ou flottant.

Pour cela, nous allons utiliser la fonction **"split()"** qui permet de couper une chaine de caractères en une liste contenant les éléments de la chaine. Il est possible de spécifier le type de séparateur et le nombre de coupes à effectuer :

Liststring = string.split(separateur, maxsplit)

Par défaut, le séparateur est l'espace entre les caractères et maxsplit = -1 (toutes les coupes possibles sont effectuées).

Quand le nombre de coupes est spécifié, le nombre d'éléments dans la liste est égale au nombre de coupes plus un élément.

Dans l'exemple de programme précédant, la partie intéressante de la chaine de caractère est la valeur de l'entrée analogique. Pour isoler cette valeur, nous allons couper la chaine en 2 éléments avec comme séparateur **" : "** grâce à l'instruction suivante :

Listdonnees = donnees.split(":",1)

Et convertir, en nombre entier, l'élément de la chaine correspondant à la valeur de A0 :

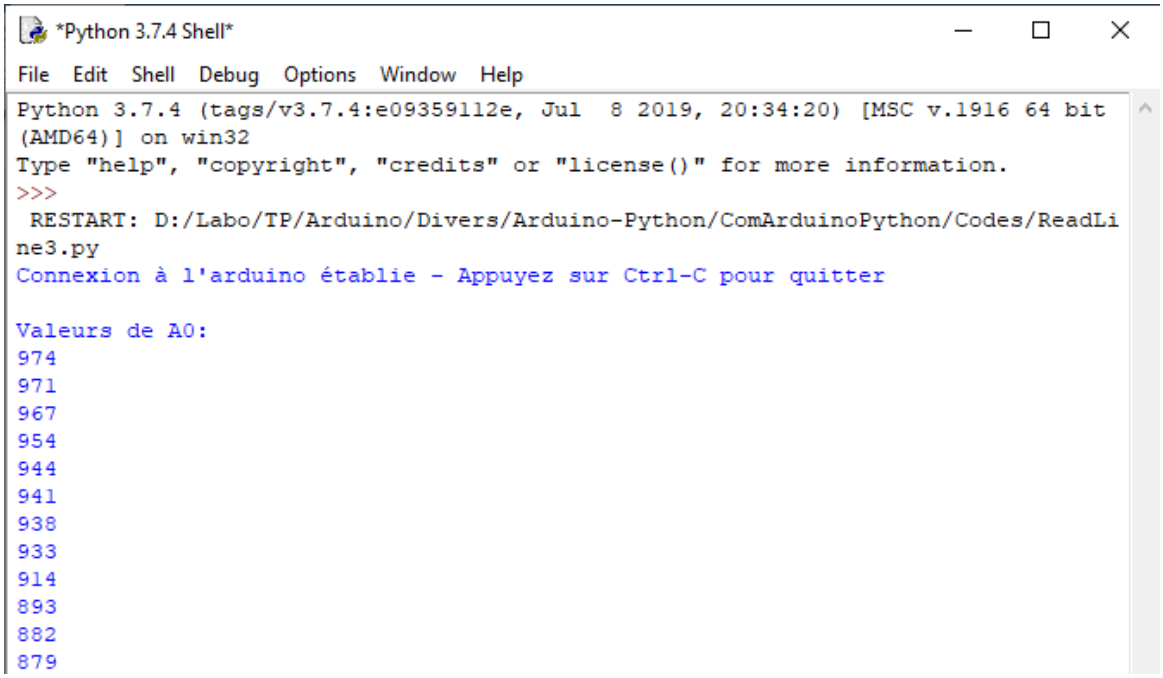
ValAO = int(Listdonnees[1])

Le premier élément de la liste **"Listdonnees"** étant : **Listdonnees[0] = "Valeur A0"**

La boucle principale du programme (programme "ReadLine3.py" dans le dossier "Codes/PY") est alors :

```
# Boucle principale du programme

print("Valeurs de A0:")

while True:

    try:
        donnees=str(SerialPort.readline())
        if donnees!="b''":
            donnees=donnees[2:len(donnees)-5]
            Listdonnees = donnees.split(":",1)
            ValA0 = int(Listdonnees[1])
            print(ValA0)

    except KeyboardInterrupt:
        SerialPort.close()
        sys.exit(0)
```

Ce qui donne dans la console Python :



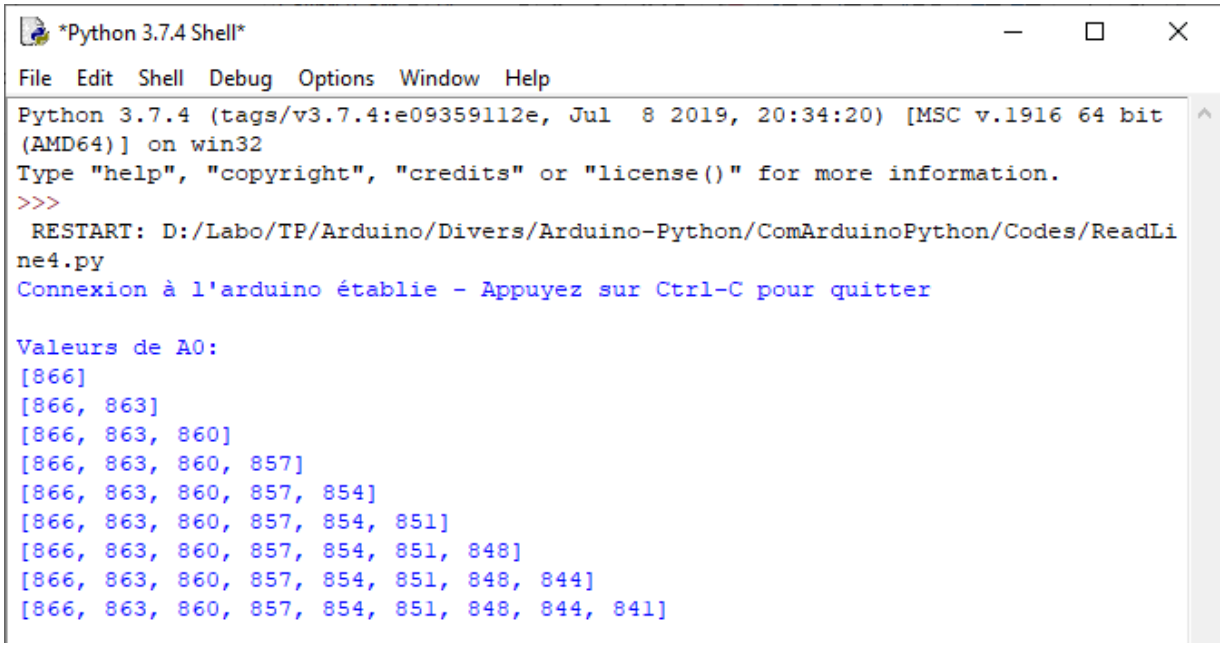
Les valeurs de A0 ont été isolées et converties en entier. Elles peuvent être maintenant stockées dans un tableau ou une liste pour effectuer ultérieurement des calculs.

Après avoir déclaré, une liste **"ListValA0 = []"**, Il suffit de remplacer le **"print(ValA0) "** par :

```
ListValA0.append(ValA0)          #ajout de ValA0 à la liste "ListValA0"
Print (ListValA0)                #pour contrôler l'ajout
```

(Voir programme "ReadLine4.py" dans le dossier "Codes/PY")

Toutes les valeurs de A0 sont alors stockées dans une liste :



2. Envoi de données par un programme Python vers l'Arduino

Pour répondre à un besoin d'information du programme téléversé dans la mémoire de l'Arduino (valeurs d'une variable, actions à effectuer…), il est parfois nécessaire que le programme Python envoie des données vers le microcontrôleur via le port série.

Pour cela, on utilise la fonction **"write()"** de l'objet **"port série**" créé au préalable :

SerialPort = serial.Serial("COM 5", baudrate=9600, timeout=1)

Généralement, les données à envoyer seront des chaines de caractères que le programme de l'Arduino pourra traiter selon les méthodes vues précédemment. Cependant la fonction **"write()"** écrit des données binaires sur le port série. Ces données sont envoyées comme une série d'octets.

Il faut donc convertir la chaine de caractères à envoyer en données binaires. On utilise pour cela la fonction **"encode('utf-8')"** qui permet de convertir la chaine de caractères en une séquence d'octets selon un encodage universel, l'UTF-8 (abréviation de l'anglais Universal Character Set Transformation Format - 8 bits), qui réunit les caractères utilisés par toutes les langues.

L'instruction d'envoi d'une chaine de caractères **"String"** est alors :

SerialPort.write(String.encode('utf-8'))

Exemple de programme d'envoi de données :

Revenons à notre programme (nommé "LedRVB.ino" dans le dossier "Codes/INO") qui sera téléversé dans la mémoire de l'Arduino et qui demande à l'utilisateur de saisir un caractère :

. '**R**', '**V**' ou '**B**' pour allumer respectivement la DEL rouge, verte ou bleue

. '**0**' pour éteindre les DELs

Le programme Python (nommé "WriteLedRVB.py" dans le dossier "Codes/PY") suivant, gère la demande de saisie de caractère et envoie la donnée à l'Arduino afin qu'il effectue l'action demandée :

```
# Programme associé avec "LedRVB.ino"

# Importation des modules

from ConnectToArduino import *

# Déclaration des constantes et variables

saisie = ''

# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
SerialPort = OpenPortCom(PortComArduino)

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


# Boucle principale du programme

print("Pour allumer la DEL rouge, envoyez: R");
print("Pour allumer la DEL verte, envoyez: V");
print("Pour allumer la DEL bleue, envoyez: B");
print("Pour éteindre les DELs, envoyez: 0\n");

while True:

    try:
        saisie = str(input(""))
        SerialPort.write(saisie.encode('utf-8'))

    except KeyboardInterrupt:
        SerialPort.close()
        sys.exit(0)
```

Jusqu'à la boucle principale, le déroulement du programme est identique à celui du programme précédent de réception de données (Importations des librairies, déclaration des variables, connexion à l'Arduino).

Après avoir affiché, les informations sur les caractères à saisir pour effectuer les actions programmé dans l'Arduino, le programme attend la saisie d'un caractère avec l'instruction :

saisie = str(input(""))

Puis envoie la donnée à l'Arduino : **SerialPort.write(saisie.encode('utf-8'))**

Ces instructions étant dans une boucle infinie, une fois la donnée envoyée, le programme attend une nouvelle saisie jusqu'à la fin du programme en appuyant sur Ctrl-C.

Le même programme python peut être utilisé avec le programme pour Arduino (nommé "ReadString.ino" dans le dossier "Codes/INO") qui, toujours avec le même circuit, va demander à l'utilisateur d'envoyer un message. Si le message est "**ON**", la DEL rouge s'allume, et si c'est "**OFF** ", la DEL rouge s'éteint.

La seule modification est sur l'information affichée précisant le message à envoyer (programme nommé "WriteLedRVB2.py" dans le dossier "Codes/PY") :

```
# 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(""))
        SerialPort.write(saisie.encode('utf-8'))

    except KeyboardInterrupt:
        SerialPort.close()
        sys.exit(0)
```

3. Réception et envoi de données simultanément par un programme Python

Les programmes Python de communication avec l'Arduino étudiés jusqu'à présent, permettaient soit de recevoir, soit d'envoyer des données. Mais pour qu'il soit complet, le programme Python doit pouvoir gérer les deux actions simultanément.

Le problème se pose quand on demande à l'utilisateur de saisir au clavier une donnée et que l'Arduino continu d'envoyer des informations. Le programme Python étant bloqué sur la saisie de l'utilisateur, les données envoyées par le microcontrôleur ne peuvent pas être lues.

Pour effectuer, l'envoi et la réception des données simultanément, il suffit de créer deux boucles, l'une pour la gestion de la réception, et l'autre pour la gestion de l'envoi.

C'est de la programmation parallèle, c'est-à-dire que plusieurs instructions de code s'exécuteront en même temps, ou presque en même temps.

Pour cela, nous allons utiliser le module "**threading**" qui permet d'exécuter un code indépendamment du code principal, ce code est appelé un "**thread**"

Avant de l'utiliser dans un programme Python, il faut bien-sûr, l'importer :

import threading

et créer une classe qui hérite de threading.Thread:

```
class MyThread(threading.Thread):
    def __init__(self, target):
        threading.Thread.__init__(self)
        self.target = target
        self.StopThread = False
    def run(self):
```

```
        while not self.StopThread:
            self.target()
    def stop(self):
        self.StopThread = True
```

Ensuite, il faut créer un objet thread :

Thread = MyThread(target)

et définir la fonction "**target**" qui est appelée au lancement du thread et contient le code qui doit s'exécuter en parallèle du reste du programme.

Le thread est lancé à l'aide de l'instruction : **Thread.start()**

La méthode "**run()**" du thread est alors appelé qui elle-même appelle la fonction "**target**" qui s'exécute en boucle tant que le thread n'est pas arrêté à la fin du programme par : **Thread.stop()**

Exemple :

Reprenons le programme ("Parseint.ino" dans le dossier "Codes/INO") à téléverser dans la mémoire de l'Arduino qui permet de régler la luminosité de la DEL rouge.

Ce programme demande à l'utilisateur de saisir la valeur souhaitée de la luminosité (nombre entier entre 1 et 255), puis renvoie la valeur choisie.

Dans le programme Python ("WriteThread.py" dans le dossier "Codes/PY") associé au programme Arduino, nous allons créer un "**thread**" pour la réception des données envoyés par l'Arduino et une boucle "**while True**" pour la saisie des valeurs de luminosité qui aura lieu après chaque réception suivant le déroulement suivant :

- Première réception : L'Arduino demande de saisir la luminosité (chaine de caractères reçue et affichée dans la console Python par le "thread")

- Saisie de la luminosité par l'utilisateur (chaine de caractères envoyée par la "boucle while True")

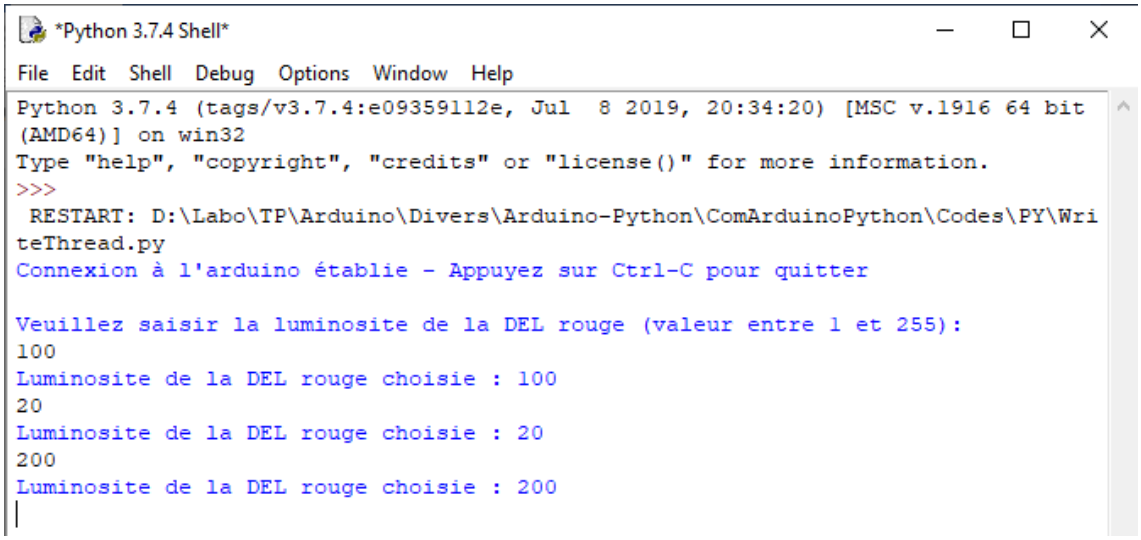
- Deuxième réception : L'Arduino renvoie la valeur choisie (nouvelle chaine de caractères reçue et affichée dans la console Python par le "thread")

- Nouvelle saisie de la luminosité par l'utilisateur (chaine de caractères envoyée par la "boucle while True")

- etc, …

```python
# Programme associé avec "Parseint.ino"

# Importations des librairies et définition des fonctions

from ConnectToArduino import *

import threading

class MyThread(threading.Thread):
    def __init__(self, target):
        threading.Thread.__init__(self)
        self.target = target
        self.StopThread = False
    def run(self):
        while not self.StopThread:
            self.target()
    def stop(self):
        self.StopThread = True

def ImportDonneeSerial():
    global DonneeRecue
    donnees=str(SerialPort.readline())
    if donnees!="b''":
        donnees=donnees[2:len(donnees)-5]
        if donnees!="":
            print(donnees)
            DonneeRecue=DonneeRecue+1

# Déclaration des constantes et variables

donnees = ''
DonneeRecue = 0

# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
SerialPort = OpenPortCom(PortComArduino)

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

# Boucle principale du programme

SerialThread = MyThread(ImportDonneeSerial)
SerialThread.start()

while True:
    try:
        if DonneeRecue==1:

            SaisieOk=False
            while SaisieOk==False:
                saisie = input("")
                try:
                    saisie=int(saisie)
                    assert saisie >= 1 and saisie <= 255
                    SaisieOk = True
                except AssertionError:
                    print("Le nombre indiqué n'est pas entre 1 et 255 !")
                    SaisieOk = False
                except:
                    print("Vous n'avez pas saisi un nombre entre 1 et 255 !")

            SerialPort.write(str(saisie).encode('utf-8'))
            DonneeRecue=0

    except KeyboardInterrupt:
        SerialThread.stop()
        SerialPort.close()
        sys.exit(0)
```

Résultats dans la console Python :



Déroulement du programme :

- Importation des librairies et définition de fonctions (idem programmes précédents avec en plus) :

. Importation de la bibliothèque "**threading**" pour la création des threads,

. Création de la classe "**MyThread**",

. Définition de la fonction "**ImportDonneeSerial()**" qui sera la fonction appelée en boucle par le thread et qui permet de recevoir les données envoyées par l'Arduino.

- Déclaration des constantes et variables :

. **donnees = ''** (variable pour stocker les données qui transitent par le port série),

. **DonneeRecue = 0** (compteur des données reçues, incrémentation de la variable dans la fonction gérant la réception des données).

- Connexion à l'Arduino (idem programmes précédents) :

Si l'ouverture du port série a réussi, on crée un objet "thread" dont l'argument est la fonction à appeler, et on le lance :

SerialThread = MyThread(ImportDonneeSerial)
SerialThread.start()

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

→ Attente de la saisie de la valeur la luminosité si la demande par l'Arduino a été envoyé et reçue **("if DonneeRecue==1"**) et envoie de la donnée à l'Arduino:

SerialPort.write(str(saisie).encode('utf-8'))

→ Réinitialisation du compteur des donnée reçues (**"DonneeRecue=0"**) si un envoi a été fait.

- Fin du programme en appuyant sur Ctrl-C :

→ Arrêt du thread en cours : **SerialThread.stop()**