Python – Acquisition de données via le port série

Port série

Un ***port série***, également appelé port COM, est un type ***d'interface informatique*** qui permet la ***communication entre un ordinateur et des périphériques externes***.
Il s'agit d'un port physique sur un ordinateur ou un appareil qui permet d'envoyer et de recevoir des données bit par bit séquentiellement sur un seul fil.

Objectif ✔ : lire un flux de données sur le port série.

En pratique 🛠 : les données sont générées par Arduino.

Prérequis 💻 : installation de la bibliothèque **pyserial**

En cas d'erreur ModuleNotFoundError: No module named 'serial' provoquée par l'instruction `import serial`, installer la bibliothèque pyserial (dans la console de Pyzo ou de Spyder, entrer : `pip install pyserial`).

Principe

1. Charger le module serial.
2. ***Définir le port série*** (ou le détecter automatiquement), ***définir la vitesse de transmission*** et ***ouvrir le port***.
3. Initialiser une variable de sortie de boucle.
4. ***Répéter*** en boucle :
   a. ***lire*** une ligne de données (les données sont au format binaire) ;
   b. ***traiter*** la ligne (suppression des caractères « invisibles » et convertir au format texte) ;
   c. ***stocker*** la ligne traitée dans une variable ou un fichier (ou simplement l'afficher en phase de test) ;
   d. si nécessaire incrémenter la variable de sortie de boucle ;
   e. tester la condition de sortie de boucle et sortir le cas échéant.
5. ***Fermer le port série***.

L'ordre des étapes dépend de la condition de sortie de boucle utilisée.

👁 Ci-contre, exemple de données envoyées par Arduino via le port série, lues ligne par ligne et affichées dans le shell python ou le moniteur série Arduino (avec émission de la chaîne `'Stop'` pour signaler la fin de la transmission de données).

```
1;1;178
2;1;182
3;1;190
…
49;10;200
50;10;200
Stop
```

⚠ **L'onglet "Serial Monitor" de l'IDE Arduino doit rester fermé pour que python puisse ouvrir le port série**.
Sous peine d'erreur python :
serial.serialutil.SerialException: could not open port 'COM3': PermissionError(13, 'Accès refusé.', None, 5)

💡 En réalité une ligne lue dans le port série contient des **caractères invisibles**, par exemple :

- les sauts de lignes codés par `'\n'` en python ;
- les tabulations (sauts de colonnes, touche tab du clavier) codés par `'\t'` en python.

💻 Ces caractères sont « invisibles » dans un éditeur de texte mais pas dans le codage des fichiers, ils doivent être supprimés.
La méthode **`strip()`** permet de supprimer les sauts de lignes.

👁 Exemple de ligne lue dans le port série :

`s = '4.51;0.7\n'` → présence d'un saut de ligne `'\n'`
`s.strip()` → suppression
Renvoie la chaîne : `'4.51;0.7'`

💻 Chaque ligne lue est une **chaîne** de caractères qui contient des données séparées par un caractère particulier appelé ***séparateur de colonnes*** (point-virgule, virgule, tabulation… ce caractère est défini par le périphérique émetteur, cf. sketch Arduino).
La découpe d'une ligne (i.e. de la chaîne associée) en "colonnes" est effectuée par la méthode **`split()`** qui renvoie une liste contenant autant de chaînes qu'il y a de colonnes dans le fichier (i.e. de valeurs sur une ligne).

👁 Exemple de ligne lue dans le port série :

`s = '4.51;0.7'` → deux valeurs séparées par un point-virgule
`s.split(';')` → « découpage » de la chaîne `s` sur le critère « point-virgule »
Renvoie la liste de chaînes (obtenue par « découpage » de la chaîne `s` sur le critère « point-virgule ») : `['4.51', '0.7']`

💡 L'objectif après lecture et traitement est de reconstituer un tableau de mesures (tableau numpy de flottants).

Code minimal - Test du port série et visualisation des données dans le shell python

Le programme suivant :
- charge le module serial (ligne 1) ;
- crée une variable pour le nom du port série nommé `'COM3'` (ligne 4) ;
- crée une variable pour la vitesse de transmission à 115200 bauds (ligne 5) ;
- ouvre le port (un objet possédant des propriétés et des méthodes est créé et stocké dans la variable `port`) (ligne 7) ;
- (il manque les initialisations éventuelles pour la condition de sortie ligne 10 : ***code à ajouter***) ;
- répète en boucle (ligne 13) :
  - lire une ligne grâce à la méthode **`readline()`** de l'objet `port` (ligne 14) ;
  - supprimer les caractères invisibles (retour à la ligne…) grâce à la méthode **`strip()`** (ligne 15) ;
  - convertir la ligne binaire en chaîne au format utf-8 grâce la méthode **`decode()`** (ligne 16) ;
  - tester la condition de sortie de boucle (ligne 17, ***code à ajouter***) et sortir le cas échéant grâce à l'instruction `break` (ligne 18) ;
  - afficher la ligne dans le shell (ou la stocker dans une variable ou un fichier) (ligne 19) ;
  - (incrémenter éventuellement la variable utilisée pour la condition de sortie de la boucle (ligne 20, ***code à ajouter***))
- ferme le port (ligne 22) ⚠ **Ne pas oublier cette instruction**.

⚠ **La vitesse de transmission du périphérique doit être la même que la vitesse de transmission définie dans python.**

💻 Code minimal pour le test de la communication série

```
 1  import serial
 2
 3  # Communication : port utilisé et vitesse de transmission
 4  port_serie = 'COM3'    # Nom du port série, cf. Arduino IDE (port sélectionné)
 5  bauds = 115200         # Vitesse de transmission, cf. sketch : Serial.begin(bauds)
 6
 7  port = serial.Serial(port_serie, bauds) # Ouverture du port série
 8
 9  # Sortie de boucle
10  ...                                  # [Sortie de boucle - Code éventuel à ajouter]
11
12  # Boucle "infinie"
13  while True:
14      ligne = port.readline()          # Lecture d'une ligne sur le port série
15      ligne = ligne.strip()            # Suppression caractères invisibles
16      ligne = ligne.decode("utf-8")    # readline -> binaire, conversion
17      if ...                           # Condition de sortie de boucle - Code à ajouter
18          break                        # Sortie de la boucle while
19      print(ligne)                     # Vérification visuelle dans le shell
20      ...                              # [Sortie de boucle - Code éventuel à ajouter]
21
22  port.close()                          # Fermeture du port série
```

Documentation : https://pyserial.readthedocs.io/en/latest/shortintro.html#opening-serial-ports

💻 **Sortie de boucle** : en pratique, il vaut mieux interrompre la boucle (sinon le port série reste ouvert ce qui empêche son utilisation par d'autres périphériques).

Exemples de sortie de boucles

Sortie après un nombre prédéfini de lignes lues :

```
 9  nbre_lignes, max_lignes = 1, 100
    ...
17      if nbre_lignes > max_lignes:
18          break
19
20      nbre_lignes += 1
```

Sortie sur un mot-clé (***ce mot-clé doit être émis par le périphérique à un moment donné, cf. sketch Arduino***) :

```
 9  Mot_cle = 'Stop'
    ...
17      if 'Stop' in ligne:
18          break
19
20      # Aucune instruction nécessaire
```

↳ **Dans la suite, on envisage la détection automatique des ports ainsi que le traitement des données et leur stockage.**

La fonction `portArduino(bauds)` (code ci-dessous) permet de détecter automatiquement les ports série disponibles, de trouver le port auquel est connecté la carte Arduino et de l'ouvrir (le paramètre `bauds` permet de définir la vitesse de transmission pour ce port).

Exemple d'affichage à l'exécution de la fonction `portArduino(bauds)` avec le paramètre `bauds` valant 115200 :

```
Liste des ports détectés
Lien série sur Bluetooth standard (COM9)
Lien série sur Bluetooth standard (COM8)
Arduino Uno (COM5)

Nom du port : COM5
Port ouvert : True
Transmission : 115200
```

💻 Code pour détection automatique des ports

```
 1 import serial
 2 import serial.tools.list_ports  # Outils pour lister les ports série
 3
 4 def portArduino(bauds):         # Fonction utilitaire pour déterminer le port série
 5     ports = list(serial.tools.list_ports.comports()) # Liste d'objets
 6     print('Liste des ports détectés')
 7     for p in ports:           # Parcours des objets
 8         print(p.description)  # Affichage d'informations avec méthodes associées
 9         if "CDC" in p.description or "Arduino" in p.description:
10             port = serial.Serial(p.device, bauds) # Ouverture du port
11
12     print('\nNom du port : ',port.name)
13     print('Port ouvert : ',port.is_open)
14     print('Transmission : ', port.baudrate)
15     return port
16
17 bauds = 115200
18 port = portArduino(bauds)      # Récupération et ouverture du port série
```

💡 Ligne 9 : CDC signifie Communications Device Class (il s'agit d'un protocole de communication utilisé en particulier avec les microcontrôleurs de type Arduino).
Ligne 18 : cette ligne est équivalente aux lignes 4,5,7 du code minimal de la page précédente.

Le code ci-dessous convent aux acquisitions rapides (i.e. avec un nombre raisonnable de valeurs) car les données sont stockées dans une liste puis dans un tableau (⚠ structures volatiles).
Pour les acquisitions longues ou lorsque les données doivent être conservées (plus prudent dans tous les cas afin d'éviter les pertes en cas de « plantage »), il faut enregistrer les données dans un fichier, cf. document « Python – Fichiers texte ».
Rq : la bibliothèque panda permet de traiter de très gros volumes de données.

Ce code est très semblable au code minimal de la page 2 (avec détection automatique des ports et arrêt de la boucle grâce à un mot-clé) : seuls les éléments nouveaux concernant le stockage et le traitement sont commentés ci-dessous.

💻 Code complet

```
 1 import serial
 2 import serial.tools.list_ports
 3
 4 def portArduino(bauds):
 5     ports = list(serial.tools.list_ports.comports())
 6     print('Liste des ports détectés')
 7     for p in ports :
 8         print(p.description)
 9         if "CDC" in p.description or "Arduino" in p.description:
10             port = serial.Serial(p.device, bauds)
11     print('\nNom du port : ',port.name)
12     print('Port ouvert : ',port.is_open)
13     print('Transmission : ', port.baudrate)
14     return port
15
16 bauds = 115200
17 port = portArduino(bauds)
18
19 # Stockage des lignes lues dans le port série dans une liste
20 mesures = [] # Liste de chaînes (cf. ci-dessous)
21
22 while True:
23     ligne = port.readline()
24     ligne = ligne.strip()
25     ligne = ligne.decode("utf-8")
26     if 'Stop' in ligne:      # Message d'arrêt défini dans sketch Arduino
27         break
28     print(ligne)
29     mesures.append(ligne)    # Rappel : ligne est de type string
30 port.close()
31
32 # Traitement des mesures : liste de chaînes -> tableau de nombres
33 # mesures = liste chaînes -> data = liste de listes de flottants
34 separateurColonnes = ';' # Cf. sketch Arduino (';' ou '\t' = tabulation...)
35 data = []                # Liste de listes de chaînes (cf. ci-dessous)
36 for s in mesures:
37     L = s.split(separateurColonnes)   # ligne = str -> liste de str = colonnes
38     data.append(L)
39 # data -> dataN = tableau numpy bidimensionnel et conversion des colonnes en float
40 dataN = np.array(data, dtype=float)
```

Stockage des lignes lues dans le port série – Lignes 20 et 29
On stocke les lignes (décodées au format chaîne) dans une liste (liste `mesures` créée à la ligne 20, remplie à la ligne 29).

Extraction, traitement et stockage des données sous forme de nombres – Lignes 34 à 40
- ✓ Ligne 34 - Définition du séparateur de colonnes (***dépend du séparateur utilisé dans le sketch Arduino***).
- ✓ Ligne 36 - Initialisation d'une liste vide `data`.
- ✓ Ligne 37 - Parcours des chaînes de la liste `mesures` pour extraire les valeurs des colonnes.
  - Ligne 38 - L'instruction `s.split(separateurColonnes)` renvoie une **liste** de chaînes stockée dans `L`.
  - Ligne 39 – Ajout de la liste `L` à la liste `data`.
- ✓ Ligne 41 – Conversion de la liste `data` en tableau numpy.

La conversion en tableau numpy permet d'effectuer des traitements mathématiques sur les colonnes du tableau obtenu.
Exemple : `x = np.sin(dataN[:,0])*3` permet de définir un nouveau tableau de valeurs à partir de la colonne d'index 0.
Il est ensuite possible de tracer des courbes.