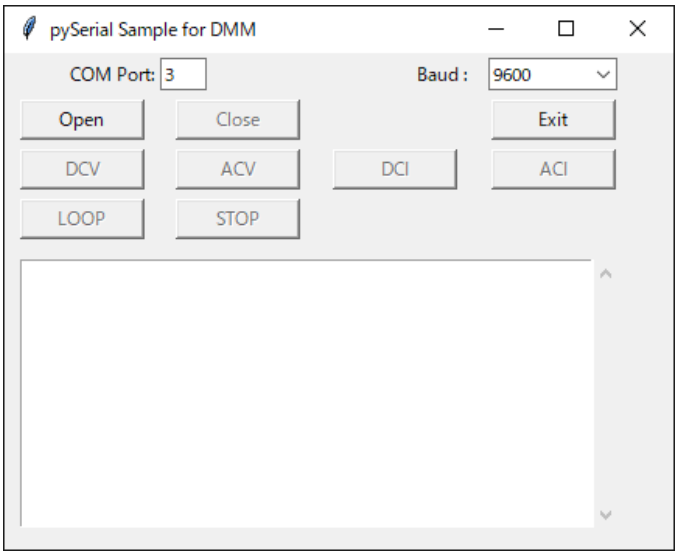
(1)　Python によるシリアル通信プログラミング

Python はプログラミングのための参考資料やサンプルが豊富にありますが、人口知能や Web アプリケーション、統計処理についてが多く、通信を使って計測機器などを制御する例はそれほど多くありません。

本ページでは GUI の作成とボタン処理を行う GUI ライブラリ(Tkinter)、通信を行う PySerial ライブラリ 、一定周期で処理を行うインターバル動作、変数の共用を行うグローバル指定を説明いたします。制御対象は弊社のマルチメーター となっていますが、文字列を変更することでシリアル通信で接続する電源やオシロスコープの他にセンサ類にも対応が可能です。

環境の準備などの手順はここでは扱いません、Python 本体の他に様々な開発ツールがありますのでインターネットで検索して python3 の実行環境をそろえてください。また、ファイル処理やグラフ出力などはインターネットに豊富にサンプルがありますので必要に応じて追加してみてください。



今回作成する Python/Tkinter での GUI 表示

Python での画面の構成としては

ボタン:9 個
ドロップダウンリスト
テキストボックス:2 個、
ラベル:2 個

で通信の OPEN/CLODE、アプリ終了、測定モードの切替えと 1 秒ごとの読取の開始と終了を割当ています。 処理としてはボタンのほかに 1 秒のインターバルでの動作も盛り込んであります。

1. 初めに必要なライブラリとグローバル変数を登録します。

```
# coding:utf-8
import serial           # pyserialモジュール
import tkinter as tk    # tkモジュール
import tkinter.scrolledtext
import time

from time import sleep
from tkinter import ttk


instr = None
Baud =['115200', '9600']
loopFlg = 0
```

ライブラリ登録は import で行います。
Python の変数は何も指定しないとローカル変数になるようなので、計測器用の変数 instr を宣言します。複数の機器を制御する場合は、instr1、instr2 などの変数を増やすことで対応します。

2. 画面のパーツを登録します。

```
root = tk.Tk()
root.geometry('430x320')
root.title('pySerial Sample for DMM')

label1 = tk.Label(root, text = 'COM Port:')
label1.grid(row=0, column=0,sticky = tk.E,pady=3)
txtPort = ttk.Entry(root , width=4 )
txtPort.delete(0,tk.END)
txtPort.insert(tk.END,'3')
txtPort.grid(row=0 , column=1,sticky = tk.W)

label2 = tk.Label(root, text = ' Baud :' )
label2.grid(row=0, column=2,sticky = tk.E)

cbDevBaud= ttk.Combobox(root , width=10 ,value = Baud )
cbDevBaud.set(Baud[1])
cbDevBaud.grid(row=0 , column=3)

txtRecive = tkinter.scrolledtext.ScrolledText(root , width=52, height=13 )
txtRecive.grid(row=4 , column=0, columnspan = 4 ,padx=10,pady=10)

btn1 = tk.Button(master=root, text='Open' , command=Open_clicked ,width=10)
btn1.grid(row = 1 , column=0,pady=3 )
btn2 = tk.Button(master=root, text='Close' , command=Close_clicked , state =tk.DISABLED,width=10 )
btn2.grid(row = 1 , column=1)

root.mainloop()
```

画面パーツは grid で配置を行っています。画面の解像度や分解能、OS の種類によってサイズが変わってしまう症状があります。複数の環境で利用する場合はかなり余裕を持った配置が必要になります。
細かいサイズ・位置を指定したい場合は place などを使用してください。
ボタンやコンボボックスなどのイベントは必要に応じてここで登録しますが、種類によって登録方法が異なります。

3. イベントの処理を登録します。

```
def Open_clicked():
    global instr

    txtRecive.delete('1.0',tk.END)

    try:
        instr = serial.Serial()
        instr.port = 'COM' + txtPort.get()        #Windows COMポート指定 Device Managerで確認
#        instr.port = 'ttyS'  + txtPort.get()     #Linux 標準COMポート  dmesgで確認
#        instr.port = 'ttyUSB'  + txtPort.get()   #Linux USB変換ポート  dmesgで確認
#        instr.port = 'ttyACM'  + txtPort.get()   #Linux USB変換ポート  dmesgで確認

        instr.baudrate =cbDevBaud.get()     # ボーレート指定
        instr.timeout = 0.5                 # タイムアウト指定

        instr.open()
        instr.write("*cls¥n".encode("utf-8"))
        instr.write("*idn?¥n".encode("utf-8"))
        sleep(0.1)
        txtRcv = instr.read(256)
        txtRecive.insert(tk.END,txtRcv.decode('ascii'))

    except:
        txtRecive.insert(tk.END,'Device Error')

def Close_clicked():
    global instr
    instr.write("syst:loc¥n".encode("utf-8"))

    instr.close()


def interval_work():                            # インターバルで行う処理を記述
    global loopFlg
    global instr

    instr.write("read?¥n".encode("utf-8"))
    sleep(0.1)
    txtRcv = instr.read(256)
    txtRecive.insert(tk.END,time.ctime() + '   ,' + txtRcv.decode('ascii').replace('¥r','')  )
    txtRecive.see('end')
    if (loopFlg == 1):
        root.after(1000, interval_work)             # インターバルの時間(ms)を指定
```

ボタンの有効・無効は['state']で切替えます。

テキストの取得は get()、削除は delete()、追加は insert ()を使用します。

指定時間後にイベントを発生させる場合は after ()を使用します、ここでは 1000ms 後に自分自身を指定しているので 1 秒ごとに処理が行われます。

デバイスを指定する instr は、それぞれの処理でグローバル宣言をしておかないと、ローカル変数が割り当てられ正しく動作しなくなりますので注意してください。

扱う文字列は計測器が ASCII ですが、python は utf-8 が標準になるため、送受信時に変換を行います。

通信設定はポート指定、ボーレート設定とタイムアウト設定を行います。ポート名は Windows では COM のみですが、Windows 以外では dmesg コマンドでポート名を確認し、読み書きの属性変更が必要なことがありますので注意して下さい。

4. まとめ

実用アプリケーションには至っていませんが、アプリの最低限の内容を取り上げました。

- ・Python3 に対応しています、Python2 で使用する場合は多少の修正が必要です。
- ・GUI は標準の Tkinter を使用し、画面配置は単純な grid を利用しました。
- ・GUI のボタンやテキストなどの最低限の扱いを記述しました。
- ・簡単なインターバル処理の方法を記述しました。
- ・通信は PySerial を使用しました。
- ・最低限のエラー処理を記述しました。
- ・グローバル変数の利用方法を記述しました。
- ・機器を増やす場合は機器のグローバル変数を追加して対応します。

後は使う方が必要なものを追加してご利用ください。

5. おまけ(Linux での認識について)

ラスベリーパイ等の Linux でシリアル通信を行う場合に、多くの製品は USB-CDC クラスのため問題なく COM ポートとして認識されますが、USB-COM 変換チップを使った製品のうち、チップに登録されたベンダ ID、プロダクト ID をオリジナルから変更した製品は Linux が正しく認識できず、dmesg コマンドでも ttyUSB または ttyACM のエントリーが表示されません。

このままで正しく認識させるにはデバイスドライバの対応が必要です。 dmesg コマンドで変換チップのメーカーを確認し、チップメーカーが提供しているデバイスドライバのソースファイルを入手し、対応するベンダ ID、プロダクト ID を追加してドライバのビルドと適用をすることで ttyUSB または ttyACM で認識されるようになります。詳細な方法はチップメーカーで配布しているドキュメントを参照してください。
また、簡易的な対応として、チップメーカーが配布している ID 書換えツールを使用しオリジナルの ID に戻す方法もありますが、こちらはメーカーとしては推奨しておりません、テスト的にご利用ください。

現在当社の製品で使用している USB 変換チップは FTDI 製、Silab 製、WCH 製の 3 種類を利用しています。

Windows10 の USB 対応一覧
https://www.texio.co.jp/news/important-info/1644/

本資料のリンク
https://www.texio.co.jp/uploads/WebExpo/Study/Study_0001/python0001.html

完成版ソースコードのダウンロード
https://www.texio.co.jp/uploads/WebExpo/Study/Study_0001/Sample_COM_DMM.zip