Tektronix®

Keithley Days

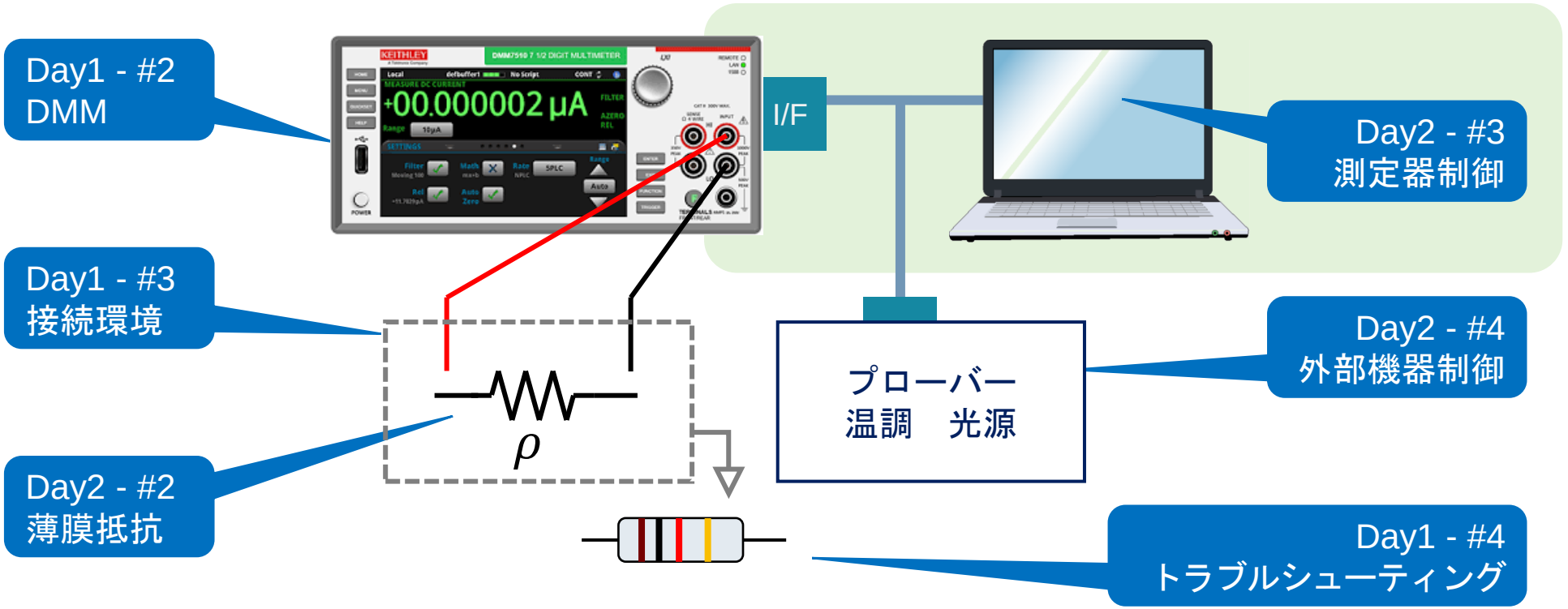
実は簡単！pythonで計測器を動かしてみる

テクトロニクス／ケースレー　アプリケーションエンジニア

桑田　祥希

KEITHLEY
A Tektronix Company

本セッションの範囲


Day1 - #2
DMM
Day1 - #3
接続環境
Day2 - #2
薄膜抵抗
I/F
Day2 - #3
測定器制御
プローバー
温調　光源
Day2 - #4
外部機器制御
Day1 - #4
トラブルシューティング
ρ

本日の内容

- １．計測器を自動制御するメリット
- ２．本セッションで扱う自動制御の範囲（PC - 計測器1台）
- ３．自動制御に共通する基礎知識
- ４．pyvisaを使い、実際に計測器を動かしてみる
- ５．KeithleyのTSPスクリプトを用いてステップアップしてみる

計測器を自動制御するメリット
何のために？

計測器を自動制御するメリット

まさにDX

「同じ手順を繰り返す」ことで、工数削減。
小さなスケールの手順 ならワンクリックで。

「判定を任せる」ことで、正確性や再現性を向上。
目や紙を使って判定しなくてもいい。

「人がいなくてもいい」ので、時間を選ばない。
休日もお昼休みも、長時間でも、データを収集。

「ネットワークを介す」ので、リモート操作できる。
家でも出張先でも誰でも、確認できる。

本セッションで扱う自動制御の範囲
小さなスケールで

本セッションにおける自動制御の範囲

小さなツールとして役に立つものを

1. 計測器自体の自動測定機能（IV sweepなど）
2. PC - 計測器1台
   フローの自動化（設定 → 測定 → 設定 → 測定 ）
   機能拡張（自動判定、アラーム）
3. PC – 計測器 n台
4. PC - 計測器 n台 – 温度計 – 光源 - - - etc.,

自動制御に共通する基礎知識
PCから装置への文字列の伝達

一般的な自動制御の道筋

PC
(OS)

プログラム

命令（“文字列”）

VISA → ドライバ → ケーブル

計測器
(Firmware)

ドライバ

プロセッサ

命令（“文字列”）

命令（“文字列”）

通信方式は、LAN, USB, GPIB, RS232C など

適切なドライバが必要

LAN, USB, GPIB, RS232など、それぞれに存在する。

ドライバがないと、プログラムが命令しても、通信に必要な物理信号を自由に生成できない。受け取っても理解できない。

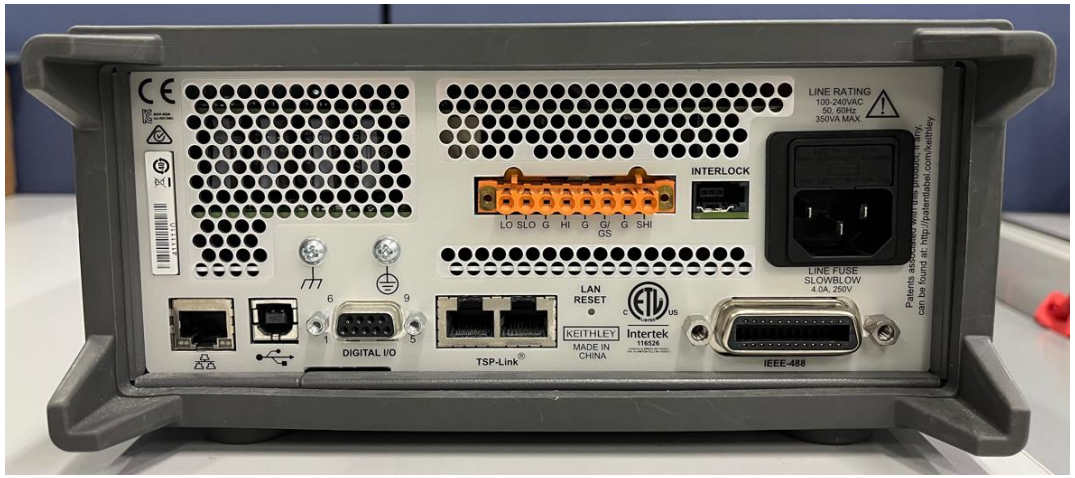
PC
(OS)

プログラム、VISA

USBでお願い

GPIBドライバ

USBはつくれません。



SMU2461型

VISA も必要

VIRTUAL INSTRUMENT SOFTWARE ARCHITECTURE、の略

プログラムから命令（”文字列”）を受け取り
適切な種類のドライバ を選んで、命令を引き渡す。その逆も。

プログラム

```
query(“*IDN?”)  #USBでもLANでも同じ書き方をさせてくれる
```

VISA

LANのためのドライバ

USBのためのドライバ

GPIBのためのドライバ

ケーブル

計測器

計測器を自動制御するとは

PCと装置の間でコマンド(“文字列”)をやり取りすること。

PC
(OS+ドライバ)

計測器
(Firmware+ドライバ)

プログラムとVISA

- Python
- C, C++
- Excel VBA
- LabVIEW
- MATLAB
- Scilab

```
write(“文字列”)
read()
query(“文字列”)
```

“*IDN?” (あなたは誰？)

“smu.source.output = smu.ON” (印加を開始して。)

“beeper.beep(1, 220)” (1秒間、220Hzで音を鳴らして。)

コマンド (“文字列”)を届ければ、計測器が動く

プログラム
- Python
- C, C++
- Excel VBA
- LabVIEW
- MATLAB
- Scilab

“smu.source.output = smu.ON” (印加を開始して。)

“beeper.beep(1, 220)” (1秒間、220Hzで音を鳴らして。)

普段はボタンやタッチだが、**“文字列”** を認識して処理を呼ぶのが、自動計測。

処理は計測器の中にたくさん準備されている。	
“smu.source.level = 1”	印加レベル設定を行う
“beeper.beep(1, 220)”	音を鳴らす

計測器におけるコマンド (“文字列”) とは

SCPIコマンド	TSPコマンド
“:SOURce:FUNCtion VOLTage” “:SOUR:VOLT 0.1” “SENSe:CURRent:RANGe 1e-6” “OUTPut ON”	“smu.source.func = smu.FUNC_DC_VOLTAGE” “smu.source.level = 0.1” “smu.measure.range = 1e-6” “smu.source.output = smu.ON”

プログラムA

```
inst_1.write("smu.source.level = 1e-6")
inst_1.write("smu.source.limit.level = 1")
```

ここが”文字列”

プログラムB

```
SMU2450.write("SOUR:VOLT 0.1")
SMU2450.write("OUTPut ON")
```

変数名(python)は好きに決める

プログラムC

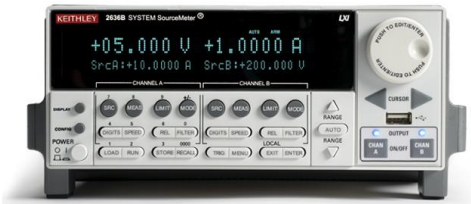
```
SMU2450.query("*IDN?”)
SMU2450.write("smu.source.level = 1e-6")
```

ここはVISA(pyvisa)

Keithleyの計測器

SourceMeter® SMU

24xxシリーズ、タッチスクリーン



26xx シリーズ

2650シリーズ、ハイパワー

Digital Multi Meter (DMM)

DMM7510型

DMM6500型

DAQ6510型

DMM+Switch

3706A型 スイッチ, DMM

Switch

707B型 スイッチングマトリクスフレーム

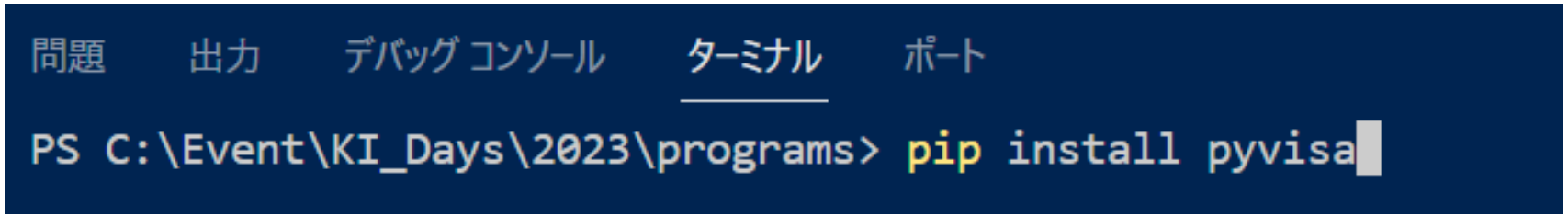
708B型 スイッチングマトリクスフレーム

pyvisaを使い、SMU2450型を動かす

pyvisaのインストール

pythonの一般的なパッケージと同じように、pipでインストールが可能です。

このパッケージを使えば、pythonでVISAの機能を簡単に実装することができます。



まずは、計測器を認識できている？

list_resources.py

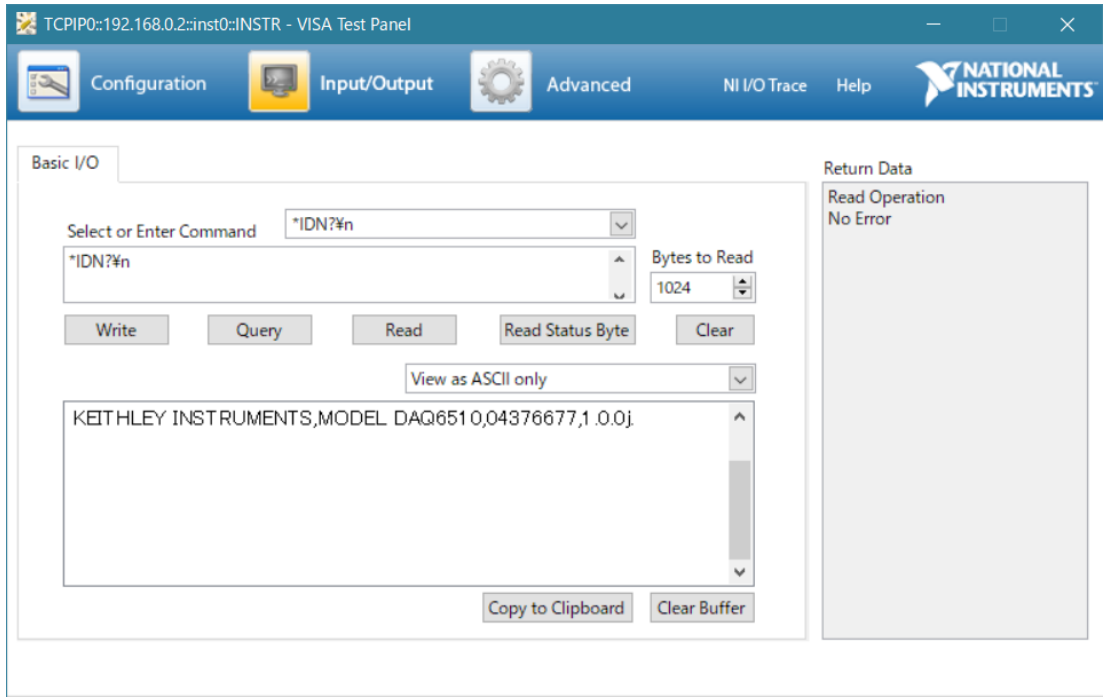
```python
# このpythonスクリプト内でpyvisaライブラリを使用可能にする。
import pyvisa
# PCと接続されている（使用可能な）リソースをrmという名前で取得。
rm = pyvisa.ResourceManager()
# rmに取得されている全てのリソース（?*::INSTRにマッチするモノ）を返す。
print(rm.list_resources())
```

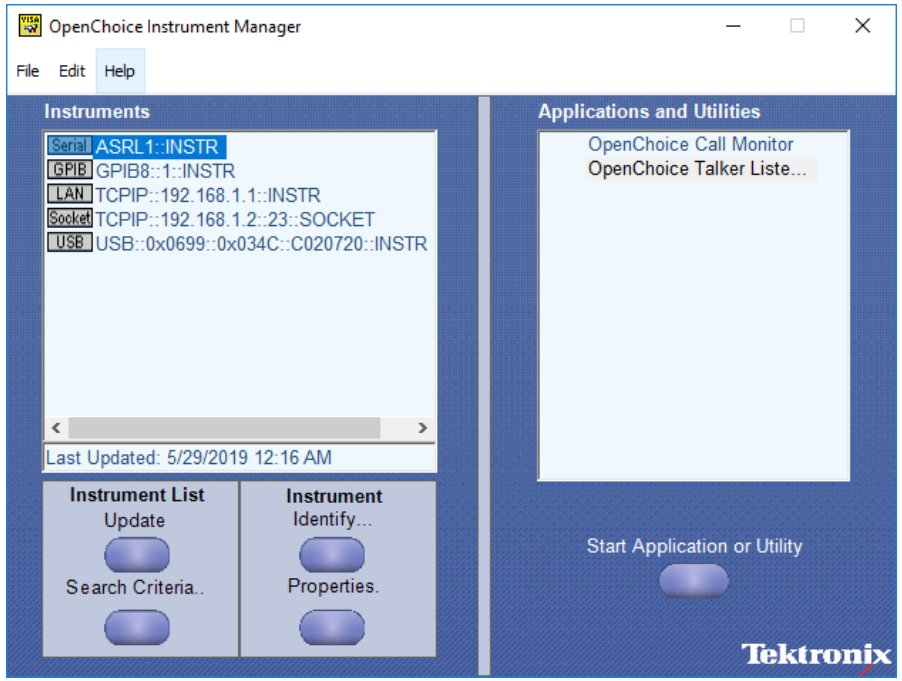
便利なツールを使用する方法も

デバイスマネージャ、VISAリソースマネージャ

少なくともどれかで、通信が確立できる状態か確認してください。

NIのリソースマネージャ、テクトロニクスOpenChoice ソフトウェア、pyvisa





簡単なことをさせてみる。ブザーを鳴らす。

idn_beep.py

```
import pyvisa
rm = pyvisa.ResourceManager()
# SMU2450という名前で、目的の計測器とコミュニケーションをopenする
SMU2450 = rm.open_resource(“USB0::0x05E6::0x2450::04509653::INSTR”)
# *IDN?をqueryしてみる。回答はanswerに。
answer = SMU2450.query(“*IDN?”)
# *IDN?の回答を表示
print(answer)
# 音を鳴らしてみる。
SMU2450.write(“beeper.beep(1, 220)”)
```

list_resources.py | idn_beep.py | reset.py | check_resistor_2.py | check_resistor_1.py

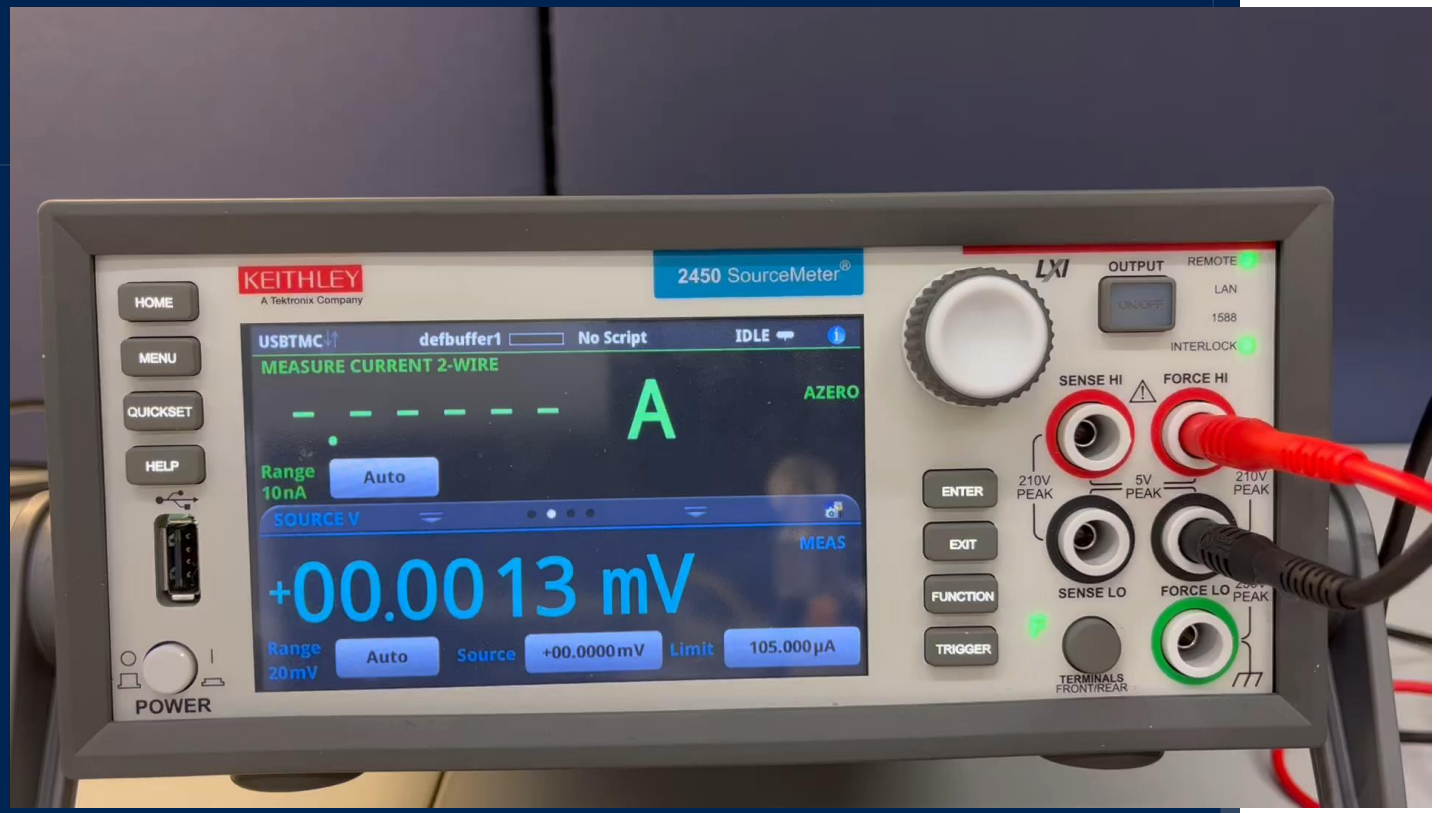
idn_beep.py > ...

```python
import pyvisa
rm = pyvisa.ResourceManager()
# inst_1という名前で、目的の計測器とコミュニケーションを開始する
SMU2450 = rm.open_resource("USB0::0x05E6::0x2450::04509653::INSTR")
# 初めてコマンドをqueryしてみる。回答はanswerに。
answer = SMU2450.query("*IDN?")
# *IDN?の回答を表示
print(answer)
# 音を鳴らしてみる。
SMU2450.write("beeper.beep(1, 220)")
```

問題 出力 デバッグ コンソール ターミナル ポート

```
PS C:\Event\KI_Days\2023\programs> python idn_beep.py
KEITHLEY INSTRUMENTS,MODEL 2450,04509653,1.7.12b

PS C:\Event\KI_Days\2023\programs>
```



行 4、列 68 スペース: 4 UTF-8 CRLF {} Python 3.6.8 64-bit

印加、測定機能の設定

check_resistor_1.py

```
rm = pyvisa.ResourceManager()
SMU2450 = rm.open_resource(“USB0::0x05E6::0x2450::04509653::INSTR”)
# smuのリセット
SMU2450.write("reset()")
# 印加機能を電流モードにして、印加レベルとリミットを設定する
SMU2450.write("smu.source.func = smu.FUNC_DC_CURRENT")
SMU2450.write("smu.source.level = 1e-6")
SMU2450.write("smu.source.limit.level = 1")
# 測定機能を電圧モードにして、測定レンジを指定する
SMU2450.write("smu.measure.func = smu.FUNC_DC_VOLTAGE")
SMU2450.write("smu.measure.range = 0.2")
```

印加についてのパラメータを設定した後、測定機能を設定する

実際に印加して測定

check_resistor_1.py

```
# 電流を実際にoutputして、測定機能を実行し、outputを止める
SMU2450.write("smu.source.output = smu.ON")
SMU2450.write("smu.measure.read()")
SMU2450.write("smu.source.output = smu.OFF")
# 測定値をバッファからqueryで取得する
voltage = float(SMU2450.query("printbuffer(1, 1, defbuffer1.readings)"))
current = float(SMU2450.query("printbuffer(1, 1, defbuffer1.sourcevalues)"))
# 抵抗値（電圧/電流）を表示する
resistance = voltage / current
print(resistance)
# SMUとのコミュニケーションをcloseする
SMU2450.close()
rm.close()
```

list_resources.py | idn_beep.py | reset.py | check_resistor_2.py | check_resistor_1.py

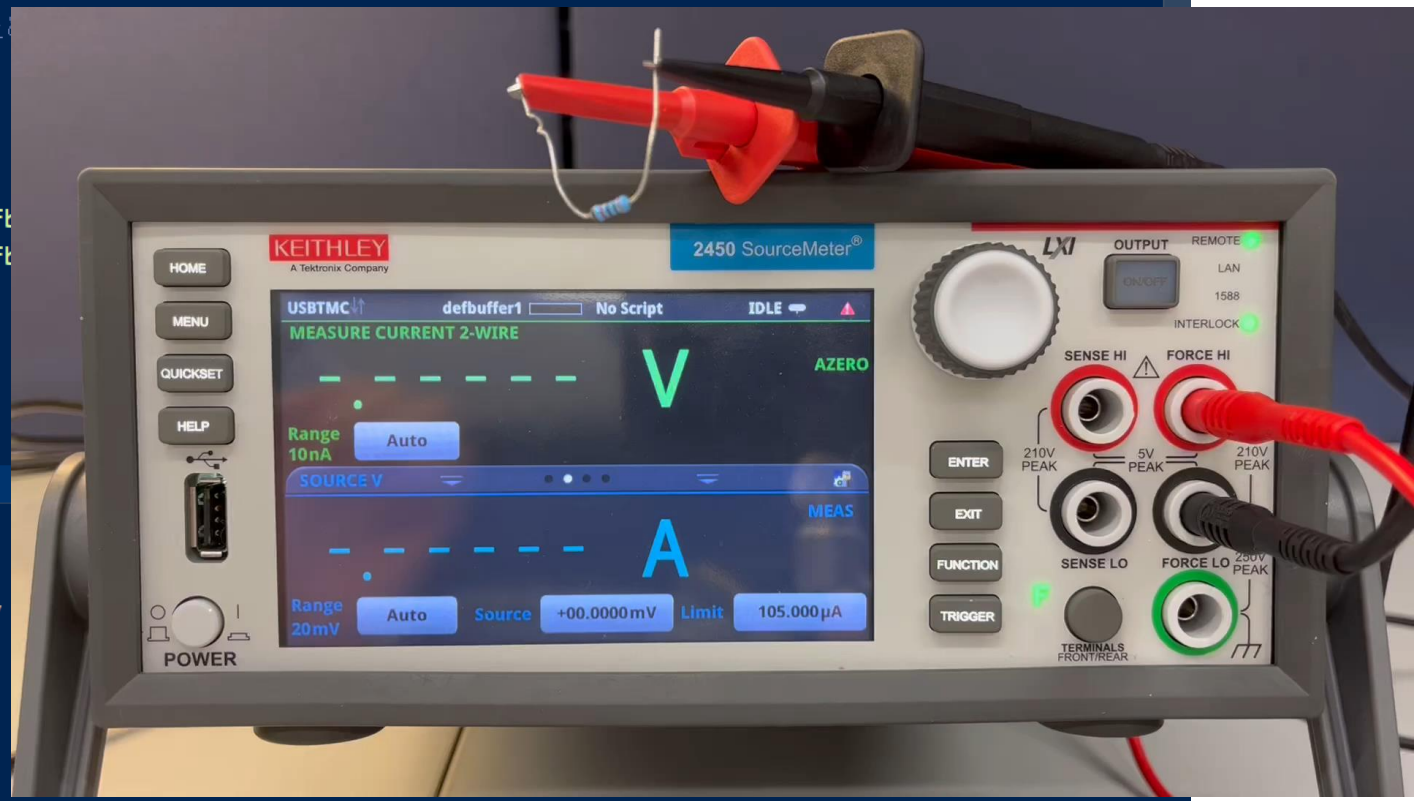
check_resistor_1.py > ...

```
5   SMU2450.write("reset()")
6   # 印加機能を電流モードにして、印加レベルとリミットを設定する。
7   SMU2450.write("smu.source.func = smu.FUNC_DC_CURRENT")
8   SMU2450.write("smu.source.level = 1e-6")
9   SMU2450.write("smu.source.limit.level = 1")
10  # 測定機能を電圧モードにして、測定レンジを指定する。
11  SMU2450.write("smu.measure.func = smu.FUNC_DC_VOLTAGE")
12  SMU2450.write("smu.measure.range = 0.2")
13  # 電流を実際にoutputして、測定機能を実行し、outputを止
14  SMU2450.write("smu.source.output = smu.ON")
15  SMU2450.write("smu.measure.read()")
16  SMU2450.write("smu.source.output = smu.OFF")
17  # 測定値をバッファからqueryで取得する
18  voltage = float(SMU2450.query("printbuffer(1, 1, defb
19  current = float(SMU2450.query("printbuffer(1, 1, defb
20  # 抵抗値（電圧/電流）を表示する。
21  resistance = voltage / current
22  print(resistance)
23  # SMUとのコミュニケーションをcloseする。
24  SMU2450.close()
25  rm.close()
```

問題　出力　デバッグ コンソール　ターミナル　ポート

```
PS C:\Event\KI_Days\2023\programs> python check_resistor_1.py
29919.777548174003
PS C:\Event\KI_Days\2023\programs>
```

有効桁にも注意



行 25、列 1　スペース: 4　UTF-8　CRLF　Python　3.6.8 64-bit

条件 (if文) を追加してみる

check_resistor_2.py

```
# 測定値をバッファからqueryで取得する
voltage = float(SMU2450.query("printbuffer(1, 1, defbuffer1.readings)"))
current = float(SMU2450.query("printbuffer(1, 1, defbuffer1.sourcevalues)"))
resistance = voltage / current
# スペックから外れていた場合は、音を鳴らす
if not 29.5e3 < resistance < 30.5e3:
    SMU2450.write("beeper.beep(0.5, 220)")
# SMUとのコミュニケーションをcloseする
SMU2450.close()
rm.close()
```

list_resources.py | idn_beep.py | reset.py | check_resistor_2.py | check_resistor_1.py

check_resistor_2.py > ...

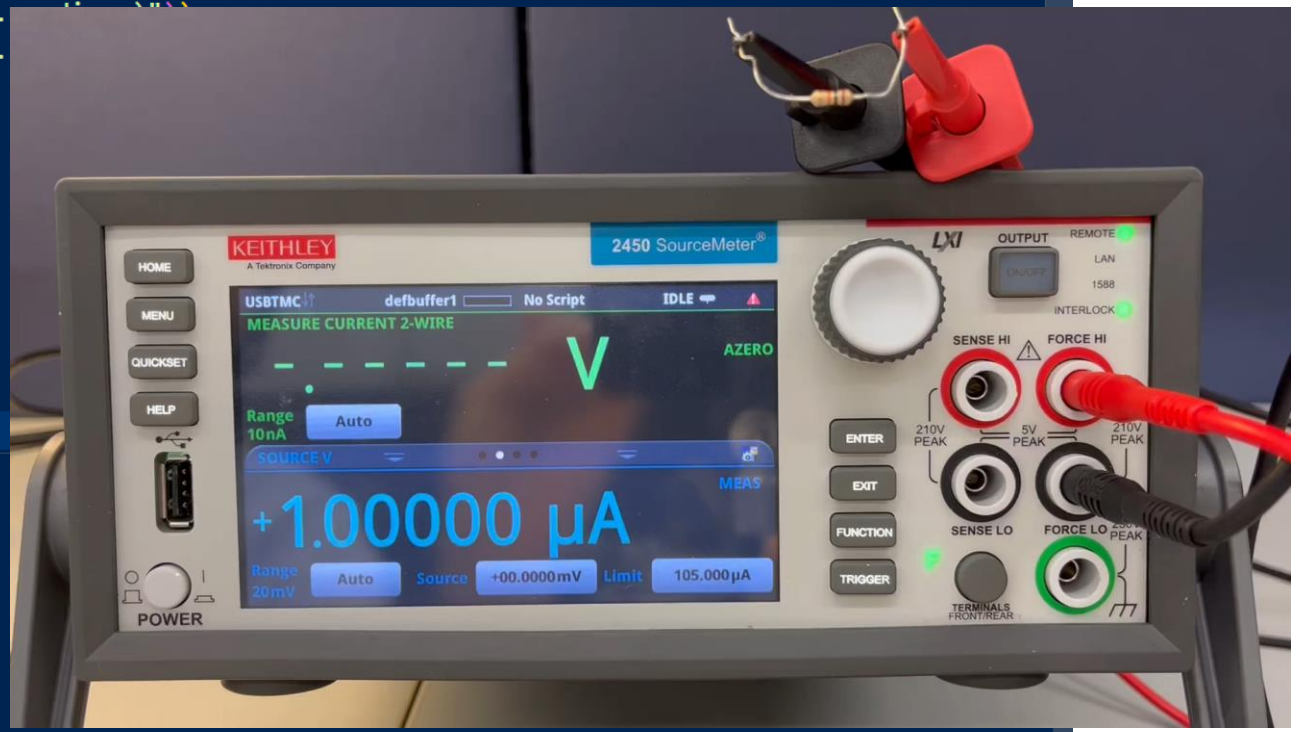
```python
SMU2450.write("smu.source.vlimit.level = 0.1")
# 測定機能を電圧モードにする
SMU2450.write("smu.measure.func = smu.FUNC_DC_VOLTAGE")
SMU2450.write("smu.measure.range = 0.2")
# 電流を実際にoutputして、測定機能を実行し、outputを止める
SMU2450.write("smu.source.output = smu.ON")
SMU2450.write("smu.measure.read()")
SMU2450.write("smu.source.output = smu.OFF")
# 測定値をバッファからqueryで取得する
voltage = float(SMU2450.query("printbuffer(1, 1, defbuffer1.
current = float(SMU2450.query("printbuffer(1, 1, defbuffer1.
resistance = voltage / current
print(resistance)

# スペックから外れていた場合は、音を鳴らす。
if not 29.5e3 < resistance < 30.5e3:
    SMU2450.write("beeper.beep(0.5, 220)")

# smuとの通信をcloseする。
SMU2450.close()
rm.close()
```

問題　出力　デバッグ コンソール　ターミナル　ポート

```
PS C:\Event\KI_Days\2023\programs> python check_resistor_2.py
9950.262782711656
PS C:\Event\KI_Days\2023\programs>
```



行 29、列 1　スペース: 4　UTF-8　CRLF　Python　3.6.8 64-bit

ここまでのメッセージ

- pyvisaを用いることで、pythonでも簡単に自動制御が可能。
  繰り返しや条件分岐のカスタマイズも容易に。
- まずは、小さなスケールの便利ツールから。
  → オペレータ業務の負担軽減、データ取りの効率化、DX。
- 自動制御の核となるのは、計測器に“文字列”を送り届けること。
  コマンドに種類はあるが、同じ文法でプログラム可能。

少しステップアップ
装置にTSPスクリプト

KeithleyのTSPスクリプト

check_resistor.tsp

```
1  --SMU2450の設定をリセット
2  reset()
3  --電流印加モードを選択し、印加レベルとリミットを設定
4  smu.source.func = smu.FUNC_DC_CURRENT
5  smu.source.level = 1e-6
6  smu.source.vlimit.level = 0.1
7  --電圧測定モードを選択し、測定レンジを設定
8  smu.measure.func = smu.FUNC_DC_VOLTAGE
9  smu.measure.range = 0.2
10 --印加を開始、測定を実行、印加を止める
11 smu.source.output = smu.ON
12 smu.measure.read()
13 smu.source.output = smu.OFF
14 --測定データを取得し、抵抗値を計算
15 voltage = defbuffer1.readings[1]
16 current = defbuffer1.sourcevalues[1]
17 resistance = voltage / current
18 --抵抗値がスペックに入っているか判定し、ダメなら音を鳴らす
19 if resistance < 29.5e3 or 30.5e3 < resistance then
20 bepper.beep(1, 220)
21 end
22
```

Keithleyの装置群を、最もシンプルにコマンド操作可能なLua言語がbaseのスクリプト言語

装置上でのワンアクションにより、一連のプロセスを実行可能

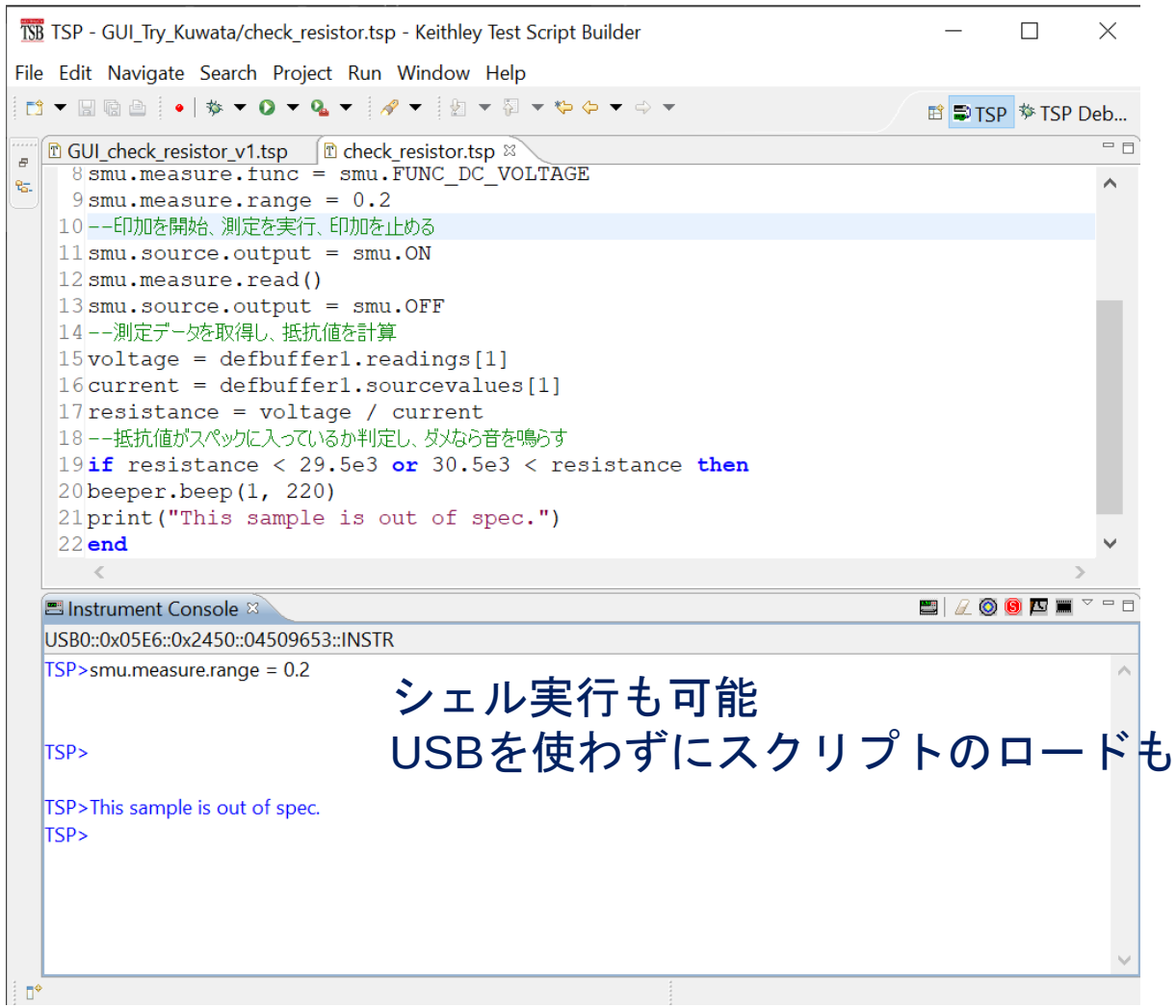
USBメモリで装置にロード
check_resistor.tsp

PC リモートで呼ぶことも可能

python

```
SMU2450.write("check_resistor()")
```

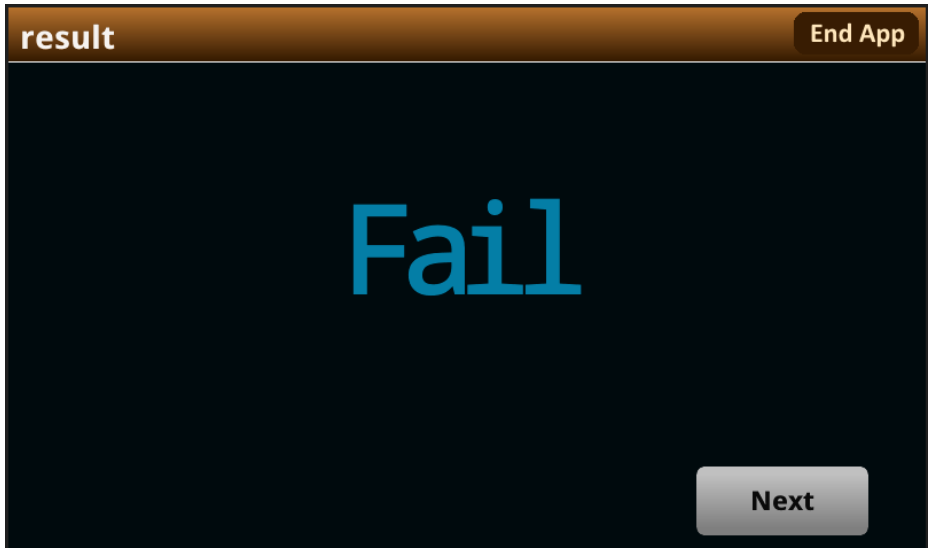
KeithleyのTest Script Builder



シェル実行も可能
USBを使わずにスクリプトのロードも可能

TSPスクリプトの作成、デバッグに最適な開発環境。無料のソフトウェア。

カスタムGUIも簡単に作成可能



参考サイト

1. Github
   https://github.com/tektronix/keithley
   https://go2.tek.com/jp-automated-measurement-lp/
2. Tektoronixの自動計測ポータル・サイト
   https://go2.tek.com/jp-automated-measurement-lp/
3. 2450 レファレンスマニュアル
   https://www.tek.com/ja/keithley-source-measure-units/keithley-smu-2400-series-sourcemeter-manual/model-2450-interactive-sou

Tektronix®

THANK YOU

ご清聴ありがとうございました

KEITHLEY
A Tektronix Company

Twitter @tektronix_jp
Facebook http://www.facebook.com/tektronix.jp