# Automate measurements with Python

Martin Mihaylov

28. September 2023

# Inhaltsverzeichnis

# 1 Introduction to the Automatic Measurments Python Libs

Automatic Measurements is a Python library dedicated to remotely controlling laboratory measurement instruments in two separate labs. The library is an ongoing project, continuously updated by me to address errors, malfunctions, and spelling mistakes as they are discovered. The library is hosted on the Uni Paderborn Git Server. To enhance functionality while ensuring code cleanliness and minimizing errors, I am currently the sole contributor responsible for submitting changes to the Git repository. However, I encourage everyone to submit suggestions and code improvements to streamline and lighten the codebase. You might wonder why Python is chosen over other programming languages like LabVIEW or MATLAB. Python leverages NiVisa, Serial Port, FPGA Control, and Socket libraries, which are also used by other programs. The advantage of Python lies in its freedom from license checkouts and package concerns. Most libraries used are free, except those provided by specific manufacturers like CST or Lumerical for EM simulations. While Python may require more lines of code, it offers a robust and license-free solution.

## 1.1 Library and Code Structures

The Automatic Measurements Library is organized into multiple folders and scripts, designed to facilitate correction, extension, and maintenance. Within each section of the code, primary functions are accompanied by informative docstrings, providing clear guidance to users. As an example, in Figure 1.1, you can observe a sample docstring.

The main.py file serves as the central hub for interconnecting all the libraries. Users can access and control all the instruments and functions through this file, which can be located directly in the Auto_Measurement folder within the Git repository. Here is a list of the instruments that can be controlled by the Python script:

1. Anrtisu spectrum Analyzer MS2760A
2. Anrtisu Signal Generator MG3694C
3. Anrtisu Vectro Analyzer MS4647B
4. Power Meter ThorLabs PM100D
5. Novoptel Laser LU1000
6. Yokogawa Optical Spectrum Analyzer AQ6370D

```
def Laserbeating_RF_Response_CoBrite(Instrument, SA_f_min, SA_f_max, Laser_f_opt, Laser_f_max, Step, resBW, LaserChannel, LaserPower, path, name, SA_
    '''


    Parameters
    ----------
    Instrument : list
        List of Instruments
    SA_f_min : float
        Spectrum Analyzer minimal frequency in Hz
    SA_f_max : float
        Spectrum Analyzer minimal frequency in Hz
    Laser_f_opt : float
        Optical frequency of the laser in THz. For Example:
            Laser_f_opt = 192.3421
    Laser_f_max : float
        Maximal frequency of the laser in THz. For Example:
            Laser_f_max = 192.3421
    Step : int
        Steps from Laser_f_opt to Laser_f_max. For Example:
            Steps = 2 , Laser_f_opt = 192.3000 Laser_f_max = 192.3400 ->
            -> (Laser_f_max- Laser_f_opt)/Steps
    resBW : float
        Resolution bandwidth in Hz
    LaserChannel : int
        Laser channel selected. It can be only 1 and 2 for CoBrite Laser Unit
    LaserPower : int
        Laser output power in dB.
    SA_TraceNum : int
        Spectrum Analyzer Active Trace. Per Defoult = 1
    path : str
        Where to save the file.
    name : str
        Name of the File. The name is Combinationof the function name and
        name that you give.
        For Example: Coupling_Stability_Test.csv

    Raises
    ------
    ValueError
        DESCRIPTION.

    Returns
    -------
    None data will be returned. A TXT File with the Power Detector data will
    be created, CSV file whit  data will be created and SVG plot
    of the measurmend will be created and saved in the given in 'path' folder.
```

***Abbildung 1.1:*** *Example of an docstring*

7. KEITHLEY Source Meter 2612
8. Power Supply KA3005 (All Programable KA  Serias Power supplys !)
9. CoBrite Tunable Laser
10. AnaPico AG,APPH20G
11. 4-Channels Power Suppy GPP4323

In the InstrumentControl folder, you'll find scripts for controlling various instruments. These scripts detail which functions of the instruments are included and which are not. Additionally, within the Auto_Measurement directory, you'll discover separate folders for each instrument, each containing a control script and a test main.py file. These main.py files were used by me when extending the functionality of the given instrument. Furthermore, you'll find PDF files containing documentation for each instrument's functions. To identify which functions are already programmed for each instrument, navigate to Auto_Measurement/Instruments Docu/_build/html/index.html. Within this directory, you can select the instrument of your choice to view the list of included functions. An example from the MS2760A can be seen in Figure 1.2.

As you can see the the instrument functions are mostly separated in two types:

1. Ask  With ask_... The user can ask the instrument for an value or data. For exmple: ask_Volt(1) Will return the Voltage measure by an GPP4323 4-Channel Power Supply

**set_Continuous**(*status*)

**Parameters:** **status** (*str/int*) – Stop/start Sweep. Can be ['ON', 'OFF', 1, 0]
**Raises:** **ValueError** – Error message
**Returns:**
**Return type:** None.

**set_ContinuousMeas**(*state*)

**Parameters:** **state** (*str/int*) – Title: Sweep Type Description: Specifies whether the sweep/measurement is triggered continuously. If the value is set to ON or 1, another sweep/measurement is triggered as soon as the current one completes. If continuous is set to OFF or 0, the instrument remains initiated until the current sweep/measurement completes, then enters the 'idle' state and waits for the :INITiate[:IMMediate] command or for :INITiate:CONTinuous ON.
**Raises:** **ValueError** – Error message
**Returns:**
**Return type:** None.

**set_DataFormat**(*status*)

**Parameters:** **status** (*str*) – Set Data Format status = ['ASCii','INTeger','REAL']
**Raises:** **ValueError** – Error message
**Returns:**
**Return type:** None.

**set_DataPointCount**(*value*)

**Parameters:** **value** (*int*) –
Title: Display Point Count
Description: Changes the number of display points the instrument currently measures. Increasing the number of display points can improve the resolution of measurements but will also increase sweep time.
Default Value: 501 Range: 10 to 10001
**Raises:** **ValueError** – Error message
**Returns:**
**Return type:** None.

***Abbildung 1.2:*** *Small cut from the index.html*

2. Set With set_... The user can set an Value or some parameter on the choosen instrument. For Example: set_VoltageLimit(A, 2.34) Will set the Voltage on Channel A to 2.34 for Keithley 2612 Power Meter.

Indeed, in addition to the standard control functions for the instruments, there are specialized functions with naming conventions like get_... or save_.... These functions are designed to extract specific data and either save it to a designated directory on your computer or make it available in your Python Variable Explorer. It's important to thoroughly review the index.html documentation to gain a comprehensive understanding of all available functions for each instrument. This will enable you to leverage the full capabilities of the Automatic Measurements Library and efficiently manage data extraction and storage.
If you have any specific questions or need further assistance with any of these functions

or instruments, please feel free to ask.

## 1.2 Set up Python

Python is readily installed on the laboratory PCs in the Optic Laboratory, complete with all the necessary libraries. To access Python, simply type Spyder into the Windows search menu. Once Spyder loads, open the main.py file and follow the instructions provided in the Python terminal. Alternatively, you can refer to Chapter 4 for guidance. Spyder is the chosen Python Integrated Development and Learning Environment (IDLE) in our laboratory. While other options like PyCharm, Vim, Visual Studio Code, and more are available, Spyder is favored for its lightweight and user-friendly nature. It can be installed as part of the Anaconda package or separately. Spyder is well-suited for scientific projects and is relatively easy to learn, particularly if you are familiar with Octave or Matlab IDLE. If Python is not installed on your PC or laptop, you can download it from Python. Ensure that you download a STABLE Python release. Following this, you can download Spyder or Anaconda from Spyder and Anaconda. You may also explore other Python APIs for your convenience.
After downloading and installing Python, you'll need to install several libraries to use the scripts. The installation process for libraries can vary depending on whether you are using Anaconda or if you installed Python first and then Spyder IDLE. If you're using Windows or Linux Terminal, you can install libraries using the following command pip install ¡Library Name¿. Most libraries can be found on PyPi or by conducting a Google search. If you're using Anaconda, you can install libraries either through the Anaconda Terminal or directly from the built-in Anaconda Library manager. Here is a list of Python libraries that were installed and used on the laboratory PC with Python version 3.6:

1. tk=8.6.10=he774522_0
2. pandas=1.1.3=py38ha925a31_0
3. numpy=1.19.2=py38hadc3359_0
4. matplotlib=3.3.2=0
5. scipy=1.5.2=py38h14eb087_0
6. seaborn=0.11.0=py_0
7. sockets=1.0.0=pypi_0
8. python-vxi11=0.9=pypi_0
9. pyusb=1.2.1=pypi_0
10. libusb=1.0.24b1=pypi_0
11. pyvisa=1.11.3=pypi_0

12. pyvisa-py=0.5.2=pypi_0
13. ftd2xx=1.1.2=pypi_0
14. ftd3xx-py3k=1.0=pypi_0
15. oct2py=5.3.0=pypi_0
16. octave-kernel=0.33.1=pypi_0

To utilize the scripts effectively, it is essential to install these libraries. While not all of them are required every time you use the scripts, having them all installed from the beginning can save you the hassle of installing new libraries each time you work with a different instrument. The numbers that follow the names of each library in the list are release and version numbers, which you can disregard. You should have at least these versions of the libraries or newer. Please be aware that this manual was written in 2023, so it's possible that some of these libraries may not be supported or could undergo changes 10 or 20 years from now. In such cases, consulting online resources and documentation or seeking assistance through online communities will be valuable for resolving any future confusion.

# 2 How to connect an instrument to PC/Laptop

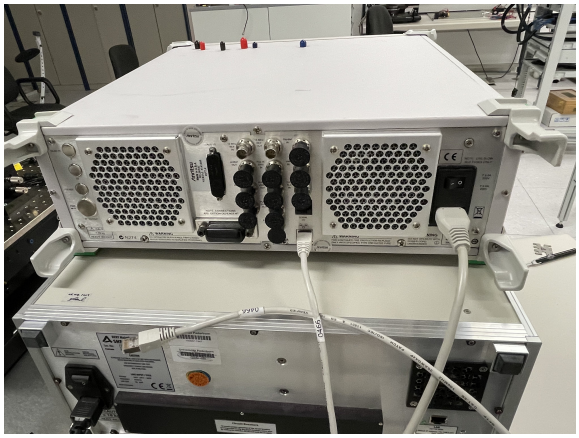
In this chapter, we will provide detailed information about the laboratory instruments and the required cables for connecting them, complete with accompanying pictures for clarity.


*(a) Anrtisu spectrum Analyzer MS2760A picture from above*


*(b) Anrtisu spectrum Analyzer MS2760A USB-C cable for PC connection*

***Abbildung 2.1:*** *Anrtisu spectrum Analyzer MS2760A picture from above 2.1a and behind 2.1b*

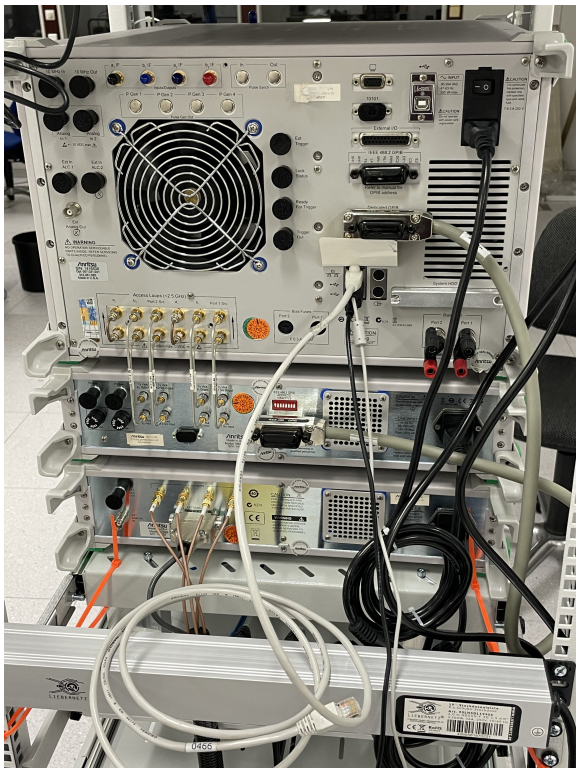

*(a) Anrtisu Signal Generator MG3694C*



*(b) Anrtisu Signal Generator MG3694C uses Ethernet cable to connect to PC*

***Abbildung 2.2:*** *Anrtisu Signal Generator MG3694C picture from above 2.2a and behind 2.2b*

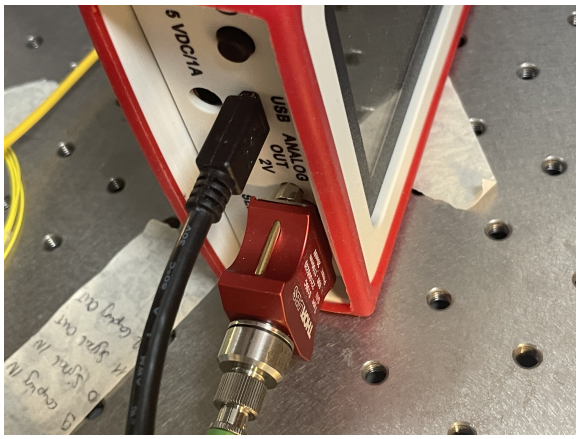

*(a) Anrtisu Vectro Analyzer MS4647B*



*(b) Anrtisu Vectro Analyzer MS4647B uses Ethernet cable to connec to PC*

***Abbildung 2.3:*** *Anrtisu spectrum Analyzer MS2760A picture from above 2.3a and behind 2.3b*

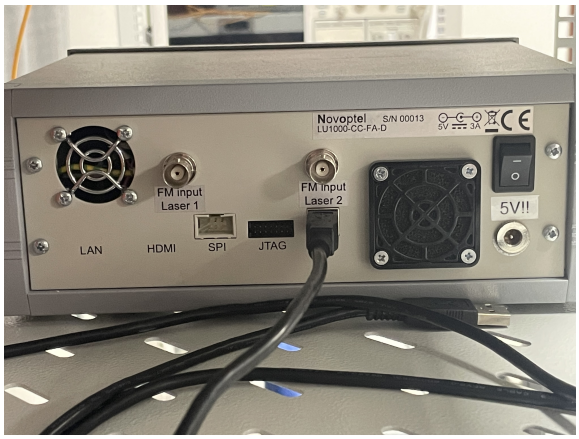

*(a) Power Meter ThorLabs PM100D*




*(b) Power Meter ThorLabs PM100D picture from the side, this instruments uses mini USB to connect to PC*

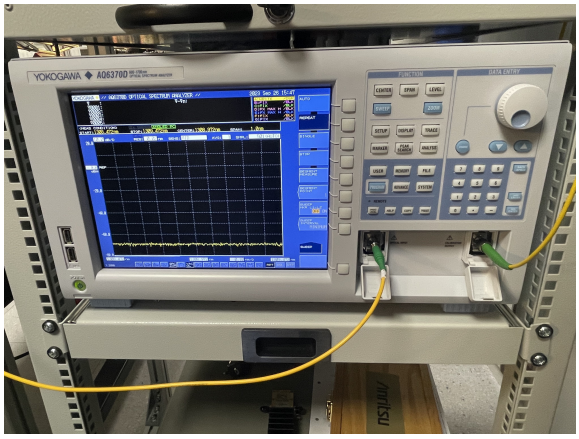
***Abbildung 2.4:*** *Power Meter ThorLabs PM100D picture from above 2.4a and the side 2.4b*


*(a) Novoptel Laser LU1000*




*(b) Novoptel Laser LU1000 uses USB-A to USB-B Cable*

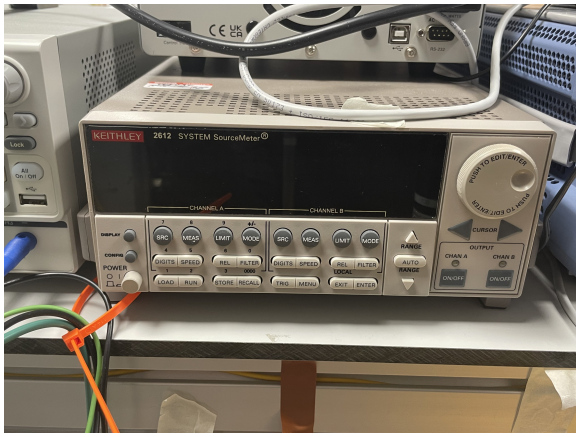
***Abbildung 2.5:*** *Novoptel Laser LU1000 front 2.5a and behind 2.5b instrument picture*



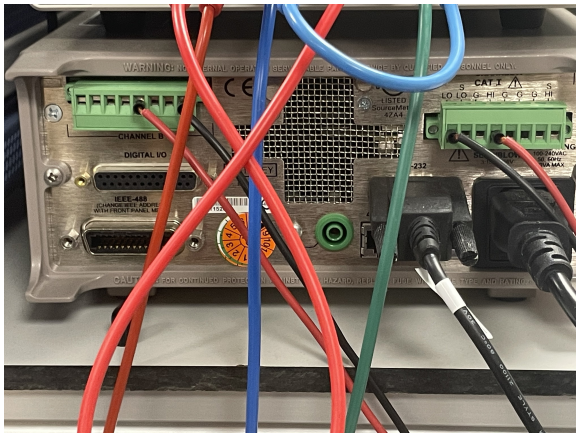

*(a) Yokogawa Optical Spectrum Analyzer AQ6370D*

*(b) Yokogawa Optical Spectrum Analyzer AQ6370D uses Ethernet cable to connect to PC*

***Abbildung 2.6:*** *Anrtisu spectrum Analyzer MS2760A front 2.3a and behind 2.3b pictures*

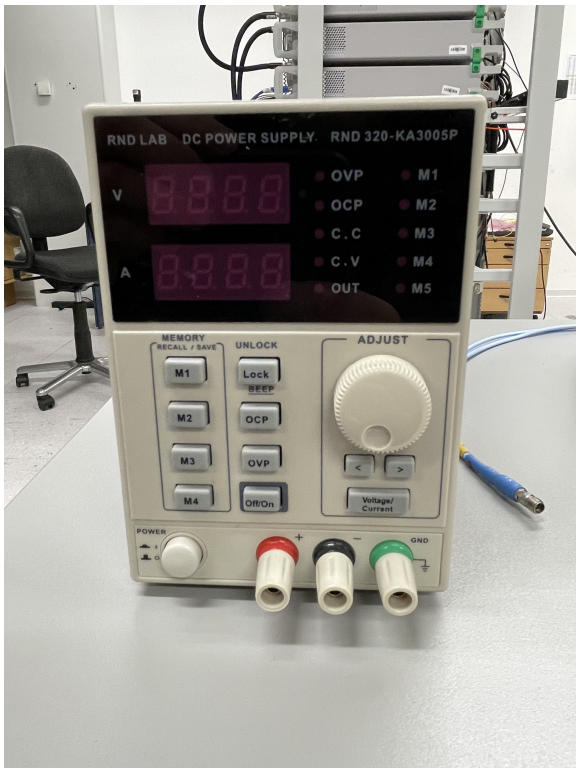



*(a) KEITHLEY Source Meter 2612*

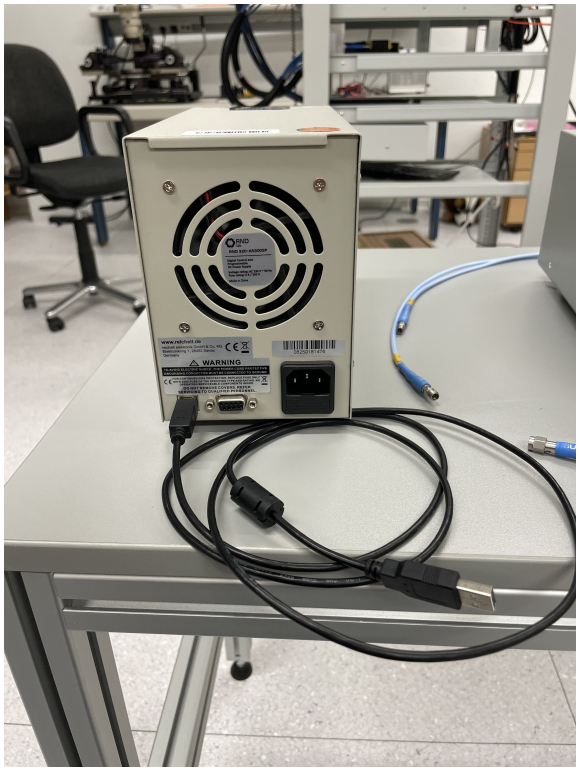



*(b) KEITHLEY Source Meter 2612 uses RS232 to USB cable to connect to PC*

***Abbildung 2.7:*** *Anrtisu spectrum Analyzer MS2760A front 2.7a and behind 2.7b pictures*



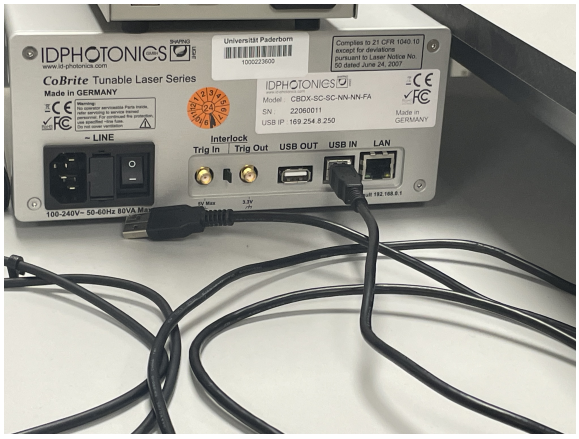

*(a) Power Supply KA3005*




*(b) Power Supply KA3005 uses USB-A to USB-B cable to connect to PC*

***Abbildung 2.8:*** *Power Supply KA3005 front 2.7a and behind 2.7b pictures*


*(a) CoBrite Tunable Laser*



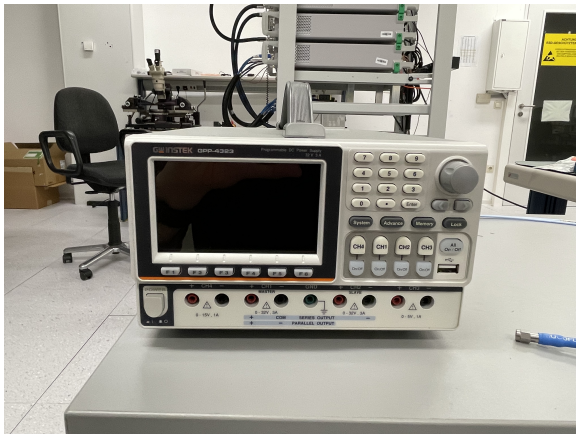

*(b) CoBrite Tunable Laser uses RS232 to USB cable to connect to PC*

***Abbildung 2.9:*** *CoBrite Tunable Laser front 2.9a and behind 2.9b pictures*

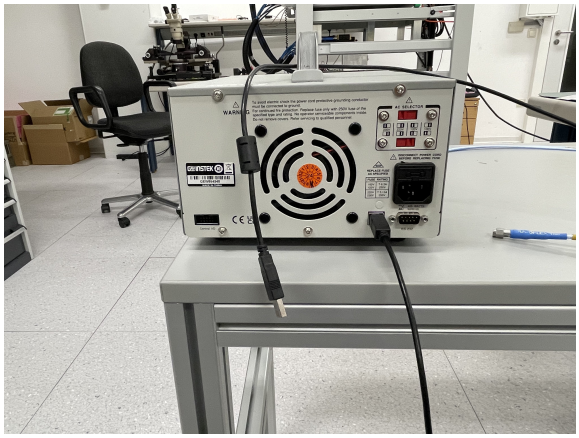

*(a) AnaPico AG APPH20G*


*(b) AnaPico AG APPH20G uses Ethernet cable to connect to PC*

***Abbildung 2.10:*** *AnaPico AG APPH20G front 2.10a and behind 2.10b pictures*



*(a) 4-Channels Power Suppy GPP4323*



*(b) 4-Channels Power Suppy GPP4323 uses USB-A to USB-B cable to connect to PC*

***Abbildung 2.11***

# 3 Script Sctructure

In this chapter the script structure will be presented. Certainly, providing a clear script structure is a valuable resource for users. It will help users navigate and understand the scripts more effectively, making it easier for them to find the information they need.
The scripts can be started in the with the main.py file. There all the needed librarys will be at first loaded. After starting the main.py the user will be askt in the python terminal to give the number of instruments that he/she want to use. For example if i have an measurments set and i need to use the power meter and voltage source i will put 2. After thus an list with all the instruments will be presented and the user need to choose witch are the two instruments. All of the instruments are numerated and the user can see the names and numbers in the python terminal. An example of the code is given in the snip below:

```
1
2  import tkinter as tk
3  from tkinter import filedialog
4  from InstrumentControl.InstrumentSelect import InstInit
5  from InstrumentControl.FunctionCall import MeasFunctions
6
7
8
9  #    ============================================================================
10 # Select Instruments and Load Instrument Libraries
11 #    ============================================================================
12
13 print('''
14       ########## How many instruments will be used ########
15
16       ''')
17 numInstr = int(input('Number Instruments = '))
18 print('''
19
20       ########## How many instruments will be used #########
21
22       ''')
23
24 listInstruments = []
25
26
27 print('''
28       ############ Select which instumnets will be used ########
```

```
29
30              You will be ask to give the name of the instruments
31              manualy. For example Instrument name: MS4647B for
32              Anrtisu Signal Analyzer.
33
34
35               Table of instruments
36               Anritsu spectrum Analyzer MS2760A = 1
37               Anritsu Signal Generator MG3694C = 2
38               Anritsu Vectro Analyzer MS4647B = 3
39               Power Meter ThorLabs PM100D = 4
40               Novoptel Laser LU1000 = 5
41               Yokogawa Optical Spectrum Analyzer AQ6370D = 6
42               KEITHLEY Source Meter 2612 = 7
43               Power Supply KA3005 = 8
44               CoBrite Tunable Laser = 9
45               AnaPico AG,APPH20G = 10
46               4-Channels Power Suppy GPP4323 = 11
47
48
49
50
51      ############ Select which instumnets will be used ########
52       ''')
53 instrumenNameList = ['1','2','3','4','5','6','7','8','9','10', '11']
54 Instrument = []
55 for i in range(numInstr):
56     instrumenName = input('Instrument name: ')
57     if instrumenName in instrumenNameList:
58         listInstruments.append(instrumenName)
59     else:
60         raise ValueError('Invalid Instrument Selected')
61
62
63 for elements in listInstruments:
64     if elements == '1':
65         SA = InstInit(elements)
66         Instrument.append(SA)
67         print('Anrtisu Spectrum Analyzer MS2760A is connected as SA')
68
69     elif elements == '2':
70         SG = InstInit(elements)
71         Instrument.append(SG)
72         print('Anrtisu Signal Generator MG3694C is connected as SG')
73
74     elif elements == '3':
75         VNA = InstInit(elements)
76         Instrument.append(VNA)
77         print('Anrtisu Vectro Network Analyzer MS4647B is connected as VNA'
    )
78
79     elif elements == '4':
80         PM = InstInit(elements)
81         Instrument.append(PM)
```

```
82          print('Power Meter ThorLabs PM100D is connected as PM')
83
84      elif elements == '5':
85          LU = InstInit(elements)
86          Instrument.append(LU)
87          print('Novoptel Laser LU1000 is connected as LU')
88
89      elif elements == '6':
90          OSA = InstInit(elements)
91          Instrument.append(OSA)
92          print('Yokogawa AQ6370D is connected as OSA')
93
94      elif elements == '7':
95          KA = InstInit(elements)
96          Instrument.append(KA)
97          print('KEITHLEY Source Meter 2612 is connected as KA')
98
99      elif elements == '8':
100         SerialNum = ['KORAD KA3005P V5.8 SN:03379314' , 'KORAD KA3005P V5.8
        SN:03379289' , 'RND 320-KA3005P V2.0']
101         PS = InstInit(elements)
102         data = PS.getIdn()
103         print(data)
104         if data in SerialNum:
105             if data == SerialNum[0]:
106                 PS1 = PS
107                 print('Power Supply KA3005P is connected as PS1')
108                 Instrument.append(PS1)
109             elif data == SerialNum[1]:
110                 PS2 = PS
111                 print('Power Supply KA3005P is connected as PS2')
112                 Instrument.append(PS2)
113             if data == SerialNum[2]:
114                 PS3 = PS
115                 print('Power Supply KA3005 is connected as PS3')
116                 Instrument.append(PS3)
117
118
119     elif elements == '9':
120         CO = InstInit(elements)
121         Instrument.append(CO)
122         print('CoBrite Tunable Laser is connected as CO')
123     elif elements == '10':
124         AP = InstInit(elements)
125         Instrument.append(AP)
126         print('AnaPico AG,APPH20G is connected as AP')
127     elif elements == '11':
128         GPP = InstInit(elements)
129         Instrument.append(GPP)
130         print('GW Instek,GPP4323 is connected as GPP')
131     else:
132         raise ValueError('Invalid Instrument Selected')
133
134
```

```
#
    ================================================================================

# Select Path for Save Data (pop up window)
#
    ================================================================================

root = tk.Tk()
path =  filedialog.askdirectory(parent = root)
root.destroy()
```

The constant prompting for commands in the terminal serves as a practical solution to the challenge of not having a graphical user interface. Once the user specifies the number of instruments they wish to use and selects the instrument(s) by name and number, the main.py script will invoke InstrumentSelect.py. Within InstrumentSelect.py, an attempt is made to establish connections with the selected instrument(s). Each instrument may require a different form of connection, such as Ethernet, USB, etc. Therefore, the script begins by inspecting the available PC ports to locate the desired instrument. To streamline the usability of the code, instruments utilizing USB connections are identified by their serial numbers. This is crucial because when a new instrument is bought for one of the laboratories, its serial number must be added to the system for user-friendly connectivity. During the process, informative messages will be displayed in the Python terminal to indicate that the system is attempting to establish connections. It's worth noting that these messages may appear multiple times if the PC has multiple USB ports. As an example, the following code snippet illustrates how power supplies may be connected within the InstrumentSelect.py script:

```
def PowerSupply():
    from InstrumentControl.RD3005 import RD3005
    from InstrumentControl.KA3005 import KA3005
    from InstrumentControl.KA3005p import KA3005p

    SerialNum = ['KORAD KA3005P V5.8 SN:03379314' , 'KORAD KA3005P V5.8 SN
    :03379289' , 'RND 320-KA3005P V2.0']
    #Prnt all instruments connected to the COM-Ports.
    #Needed to set later
    import serial.tools.list_ports
    ports = serial.tools.list_ports.comports()
    COM_List = []
    Port_ = None
    for port, desc, hwid in sorted(ports):
            # print("{}: {} [{}]".format(port, desc, hwid))
            COM_List.append(port)


    PowerInstr = 0
    for data in list(COM_List):
        while PowerInstr == 0:
            try:
                PS = RD3005(data)
                PowerInstr = PS.getIdn().split("\n")[0]
                if PowerInstr in SerialNum:
```

```
                    PowerInstr = 1
                    Port_ = data
                    break
                else:
                    PowerInstr = 0
                    print("Scanning COM Ports for Instrument !")
            except serial.SerialException as e:
            #There is no new data from serial port
                print("Scanning COM Ports for Instrument !")
            except TypeError as e:
                #Disconnect of USB->UART occured
                print("Scanning COM Ports for Instrument !")
            except visa.VisaIOError as e:
                print("Scanning COM Ports for Instrument !")
            except AttributeError:
                pass
            else:
                break
            break
    CheckInstrName = None
    CheckInstrName = PS.getIdn().split("\n")[0]
    PS.Close()
```

Äfter successfully connecting the instrument, it is passed back to the main.py script. In the main.py terminal, you will receive a notification like: Power Supply KA3005P is connected as PS1. This notification confirms the successful connection of, for example, the Power Supply KA3005P, which is identified as 'PS1.' Once the connection process for all selected instruments is completed, a final dialog window will appear, prompting you to specify the directory where you intend to save measurement data. Subsequently, you can easily interact with the instruments by using commands such as: PS1.¡Instrument function¿. In addition, the Python scripts incorporate a secondary section where predefined measurement functions are available. These functions can be initiated with:

```
MeasFunctions(Instrument,path)
```

In this section, we provide a detailed list of all the pre-coded functions available for your convenience. After selecting a function in the Python terminal, you will be prompted to provide the initial parameters required to perform measurements. You can refer to Chapter 4 for a practical demonstration of how this process works.The functions are selected in FunctionCall.py, where each function is accompanied by a docstring that provides all the necessary information. This information can be easily accessed and displayed in the Python terminal. The following code snippet illustrates this approach:

```

import pandas as pd
from InstrumentControl.Functions import Coupling_Stability
from InstrumentControl.Functions import MZM_transfer
from InstrumentControl.Functions import GC_transfer
from InstrumentControl.Functions import SG_SA_transfer
from InstrumentControl.Functions import SG_SA_Linearing
from InstrumentControl.Functions import Laserbeating_RF_Response
from InstrumentControl.Functions import Device_characterization
from InstrumentControl.Functions import MZM_transfer_Keithley
```

```
from InstrumentControl.Functions import Voltage_OSA
from InstrumentControl.Functions import Sweep_CWF_VNA
from InstrumentControl.Functions import Sweep_Power_VNA
from InstrumentControl.Functions import Laserbeating_RF_Response_CoBrite
from InstrumentControl.Functions import Tobias_MZM_passive_Equalizer
from InstrumentControl.Functions import Tobias_MZM_wo_Equalizer
from InstrumentControl.Functions import Tobias_MZM_measurement
from InstrumentControl.Functions import
    Tobias_MZM_characteristic_in_dependency_of_thermal_biasing
import os




def __init__():
    print('''
          ########## Short Function Description ##########
              1) Coupling_Stability:
                    In this function the coupling stability over time will
    be
                    measured. The Power Meter is used.

              2) MZM_transfer:
                    In this function the transfer characteristic of a MZM
    Interferometer
                    will be measured. For the purpase the MZM will be
    connected
                    to a Power Supply that will sleep the Voltage from 0V
    to a
                    given voltage. The Output of the MZM will be measured
    whit
                    the Power Meter.

              3) MZM_transfer_Keithley:
                    In this function the transfer characteristic of a MZM
    Interferometer
                    will be measured. Same as function 2) but with the
    helpt of
                    KEITHLEY Source Meter.

              4) GC_transfer:
                    In this function the LU1000 from Novoptel will be
    sweept. The
                    user will be askt to set a min and max wavelenght for
    the sweep.
                    The Power Meter will measure the output of the Laser.
    It can be used
                    to characterise different optical DUTs.

              5) SG_SA_transfer:
                    In this function the CW Frequency of the Signal
    Generator will be
                    sweept and the Spectrum will be measured with the
    Spectrum Analyzer.
```

```
                    The Spectrum Analyser is still not working properly so this function
                    is still under construction!

            6) SG_SA_Linearing:
                    This function will be used to measure and characterize non-linearitys
                    from the system(DUT). The Signal Generator will be set at fixed CW
                    frequency. The power of the Signal Generator will be sweeped. The
                    Spectrum Analyser will measure the first, secound thurd and forth
                    peaks in a given frequency range. Afterwords a Total Harminic Distrotion
                    calculation will be done and the results will be saved.

            7) Laserbeating_RF_Response:
                    Durch die Schwebung zweier Laser kann man sehr hohe elektrische
                    Frequenzen erzeugen. Zus tzlich ist die Schwebung sehr stabil in
                    der optischen Amplitude  ber das komplette C-Band. Allerdings ist
                    die Schwebungsfrequenz nicht so stabil. Deshalb sollten die Laser auf
                     Whispermode  gestellt werden. Mit dieser Messroutine kann man schnell
                    die Tranfercharakteristik von opto-elektrischen Empf ngern bestimmen.

            8) Device_characterization:
                    This function will use the Power Meter, Vectro Network Analyzer and
                    Source Meter to produce a dataset that will be used for further
                    characterisation ot better understanding of a DUT.
                    Durch diese Routine k nnen wir die Modellierung von Komponenten mit
                    Daten f ttern und bessere Simulationsmodelle bekommen.

            9) Voltage_OSA:
                    This function will use the Optical Spectrum Analyzer and the Source Meter
                    to sweep a given voltage and observe the optical signal spectrum.

            10) Sweep_CWF_VNA:
                    Sweep the continuous waveform(CW) Frequency from the Vectro analyzer Port
                    by constant outpput Power. Spectrum Analyzer will detect the
                    Signal and save the data in CSV, param and svg files.

```

```
87          11) Sweep_Power_VNA:
88                  Sweep the Power from the Vectro Analyzer Port by
    constant continuous
89                  waveform(CW) Frequency. Spectrum Analyzer will detect
    the
90                  Signal and save the data in CSV, param and svg files.
91
92          12) Laserbeating_RF_Response_CoBrite:
93                  Same as Laserbeating_RF_Response! Will use the CoBrite
    Laser
94                  and the spectrum analyzer. No pewer meter needet.
95
96          13) Tobias_MZM_passive_Equalizer:
97                  A Function that is used to measure Tobias Chip(MZM
    passive Equalizer). The instruments needed are
98                  VNA, Power Mether ThorLabs, Source Meter and Power
    Supply. The Voltage from
99                  the Power Supply will be sweeped. At each iteration,
    the source meter voltage
100                 is swept and for each voltage swept, a voltage
    measurement is made by the source
101                 meter, a power measurement is made by the power meter,
    and an S-parameter 4-port
102                 measurement is made. The information will be saved in .
    csv , .svg Files and additional
103                 .txt Files with speacial synthax suited for ADS.
104
105         14) Tobias_MZM_wo_Equalizer:
106                 A Function that is used to measure Tobias Chip(MZM wo
    Equalizer). The instruments needed are
107                 VNA, Power Mether ThorLabs, Source Meter and Power
    Supply. The Voltage from
108                 the Power Supply will be sweeped. At each iteration,
    the source meter voltage
109                 is swept and for each voltage swept, a voltage
    measurement is made by the source
110                 meter, a power measurement is made by the power meter,
    and an S-parameter 4-port
111                 measurement is made. The information will be saved in .
    csv , .svg Files and additional
112                 .txt Files with speacial synthax suited for ADS.
113
114         15) Tobias_MZM_measurement:
115                 A function which is used to measure an MZM. The
    instruments needed are
116                 VNA (Anritsu MS4647B with extension), optical
    powermeter (ThorLabs S155C),
117                 SystemSourceMeter (Keithley 2612) and 2x Power Supply (
    RND Labs 320-KA3005P).
118                 In a first step, the voltage from the first power
    supply (connected to th.PS p)
119                 is swept, while the other power supply (connected to th
    .PS n) is turned off
120                 and the optical power is measured by means of the
```

```
    powermeter.
                    This process is repeated the other way around.
                    Afterwards, the maximum, minimum and the quadrature
    points are determined and saved,
                    and the MZM is biased to quadrature point.
                    Then, either the voltage or the current of the
    SystemSourceMeter is swept.
                    Both channels are swept with same voltage or current,
                    and the corresponding current or voltage of both
    channels is measured.
                    For each voltage or current step, S-parameters are
    measured by means of the VNA.
                    All measurement results are saved in .snp, .csv, .txt
    and .svg files.
                    The syntax of the .txt files is adapted to ADS.

            16) Tobias_MZM_characteristic_in_dependency_of_thermal_biasing:
                    A function which is used to measure an MZM. The
    instruments needed are
                    VNA (Anritsu MS4647B with extension), optical
    powermeter (ThorLabs S155C),
                    SystemSourceMeter (Keithley 2612) and 2x Power Supply (
    RND Labs 320-KA3005P).
                    The power supplies deliver the voltages fed to the
    thermal phaseshifters.
                    These voltages are swept and in each step the MZM
    characteristic is measured via the
                    the VNA and the SystemSourceMeter.
                    This is repeated for multiple biasing voltages/currents
                    of the regular Si phaseshifters in the MZM.
                    All measurement results are saved in .snp, .csv, .txt
    and .svg files.
                    The syntax of the .txt files is adapted to ADS.

          ''')


def MeasFunctions(Instrument,path):
    print('''
          ############ Functions  Menu ############
              0) For short function description = 0
              1) Coupling_Stability = 1
              2) MZM_transfer = 2
              3) MZM_transfer_Keithley = 3
              4) GC_transfer = 4
              5) SG_SA_transfer = 5
              6) SG_SA_Linearing = 6
              7) Laserbeating_RF_Response = 7
              8) Device_characterization = 8
              9) Voltage_OSA = 9
              10) Sweep_CWF_VNA = 10
              11) Sweep_Power_VNA = 11
              12) Laserbeating_RF_Response_CoBrite = 12
              13) Tobias_MZM_passive_Equalizer = 13
```

```
163             14) Tobias_MZM_wo_Equalizer = 14
164             15) Tobias_MZM_measurement = 15
165             16) Tobias_MZM_characteristic_in_dependency_of_thermal_biasing = 16
166             17) Exit
167
168         ########## Functions  Menu ##########
169
170           ''')
171
172     FunctNameList = ['0','1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17']
173     FunctName = input('Function name: ')
174     if FunctName in FunctNameList:
175         while FunctName != '17':
176
177
178             if FunctName == '1':
179                 print('''
180                         Parameters
181                         ----------
182                         Time : int/float
183                             Time in minutes. How many Min the Power Detector will measure the
184                             Power.
185                         WaveLength : int
186                             Wavelenght for the Power Meter. It is give in nm. For example WaveLength = 1310, WaveLength = 1550
187                         path : str
188                             Where to save the file.
189                         name : str
190                             Name of the File. The name is Combinationof the function name and
191                             name that you give.
192                             For Example: Coupling_Stability_Test.csv
193
194
195                         Returns
196                         -------
197                         None data will be returned. A TXT File with the Power Detector data will
198                         be created, CSV file whit  data will be created and SVG plot
199                         of the measurmend will be created and saved in the given in 'path' folder.
200                       ''')
201                 print('\n')
202                 Time = float(input('Time = '))
203                 WaveLength = float(input('Power Meter Wavelenth [nm] = '))
204                 name = input('name = ')
205
206                 Coupling_Stability(Instrument ,Time, WaveLength ,path ,name)
207                 break
```

While the function descriptions are not yet complete due to ongoing work on other tasks, they are designed to be explicit enough to provide a clear idea of their intended use. Once you select a function and provide the necessary instrument and measurement parameters in the Python terminal, the information is then passed to Functions.py. In this script, you can find detailed implementations of the measurements. For instance, let's consider an example of a coupling stability measurement. The following code snippet outlines the process:

```
1
2  import numpy as np
3  import pandas as pd
4  import tkinter as tk
5  from tkinter import filedialog
6  import matplotlib.pyplot as plt
7  plt.rcParams.update({'font.size':22})
8  plt.rcParams["figure.figsize"] = (22,20)
9  import time as t
10 import sys
11 import pyvisa as visa
12 import os
13 from datetime import datetime
14
15
16
17 #    ====================================================================================

18 # Loading Bar
19 #    ====================================================================================


20
21 def loadingBar(count,total,size):
22     percent = float(count)/float(total)*100
23     sys.stdout.write("\r" + str(int(count)).rjust(3,'0')+"/"+
24                      str(int(total)).rjust(3,'0') +' [' + '='*int(percent
    /10)*
25                      size + ' '*(10-int(percent/10))*size + ']')
26
27 def loadingBarTwo(count,total,size):
28     percent = float(count)/float(total)*100
29     sys.stdout.write("\r" + str(int(count)).rjust(3,'0')+"/"+
30                      str(int(total)).rjust(3,'0') +' [' + '*'*int(percent
    /10)*
31                      size + ' '*(10-int(percent/10))*size + ']')
32
33 #    ====================================================================================

34 # Definitions
35 #    ====================================================================================
```

```
36
37
38 def Coupling_Stability(Instrument,Time, WaveLength, path,name):
39     '''
40
41
42     Parameters
43     ----------
44     Time : int
45         Time in minutes. How many Min the Power Detector will measure the
46         Power.
47     WaveLength : int
48         Wavelenght for the Power Meter. It is give in nm. For example
       WaveLength = 1310, WaveLength = 1550
49     path : str
50         Where to save the file.
51     name : str
52         Name of the File. The name is Combinationof the function name and
53         name that you give.
54         For Example: Coupling_Stability_Test.csv
55
56
57     Returns
58     -------
59     None data will be returned. A TXT File with the Power Detector data
       will
60     be created, CSV file whit  data will be created and SVG plot
61     of the measurmend will be created and saved in the given in 'path'
       folder.
62
63     '''
64     for i in range(len(Instrument)):
65         if 'PM100D' in str(Instrument[i]).split('.'):
66             PM = Instrument[i]
67         else:
68             raise Exception("""
69                                 The Instrument that you select is not one
       of:
70                                     - Power Meter
71                                 """)
72
73     name = 'Coupling_Stability_' + name
74     nameParam = name + '_Param'
75     nameImage = name + '_Image'
76
77     secs = Time*60
78     TimeVec = np.arange(1,int(secs+1),1)
79     PowerVec = []
80     Headaer = ['Time/s','Power/'+PM.ask_PowerUnits()]
81     for i in range(secs):
82         loadingBar(i,int(secs+1),1)
83         t.sleep(1)
84         PowerVec.append(PM.DefaultPowerMeas(WaveLength))
85     PowerVec = np.array(PowerVec, dtype=np.float32)
```

```
    print('Max Power = ',max(PowerVec))
    print('Min Power = ',min(PowerVec))
    data = {Headaer[0]:TimeVec,Headaer[1]:PowerVec}
    PD = pd.DataFrame(data,columns = Headaer)
    PD.to_csv(path +'/'+ name +'.csv',sep = ',')
    Headers,Data,Param = PM.DisplayParamDict('Power')
    ParamsInst = {}
    for i in range(len(Param)):
        ParamsInst[Param[i]] = Data[i]
    with open(path +'/'+ nameParam +'.txt', 'w') as file:
        for key, value in ParamsInst.items():
            file.write(key+"\t"+value+"\n")

    plt.figure()
    plt.plot(PD[Headaer[0]],PD[Headaer[1]])
    plt.xlabel(Headaer[0])
    plt.ylabel(Headaer[1])
    plt.title(name)
    plt.grid()
    figManager = plt.get_current_fig_manager()
    figManager.window.showMaximized()
    plt.show()
    plt.savefig(path +'/'+ nameImage +".svg")
```

After completing your measurements, the function will return the measured data in two formats: an .svg image visualizing the data and the raw data saved in either .txt or .csv format and an .txt file with all the instrument settings for the specific measurment. All of this data will be conveniently stored in the directory you specified at the beginning. However, it is imperative to close all the communication ports that were opened between your PC/Laptop and the instruments. Failing to do so may lead to issues. You can achieve this for each instrument in the main.py file using the following code snippet:

```
#dont forget to close the instrument connection to the PC. (free the socket)

# To close Anrtisu spectrum Analyzer MS2760A
SA.Close()

# To close Anrtisu Signal Generator MG3694C
SG.Close()

# To close Anrtisu Vectro Analyzer MS4647B
VNA.RTL() # Disconnect from VNA so that you can use the VNA afterwards. If not you need to restart the VNA!!!
VNA.Close()

# To close Novoptel Laser LU1000
LU.Close()

# To close  Power Meter ThorLabs PM100D
PM.Close()

# To close KEITHLEY Source Meter 2612
KA.Close()
```

```
21
22 # To close Power Supply KA3005 All Programable KA
23 PS.Close()
24 PS1.Close()
25 PS2.Close()
26 PS3.Close()
27
28 # To close Yokogawa Optical Spectrum Analyzer AQ6370D
29 OSA.Close()
30
31 # To close CoBrite Tunable Laser
32 CO.Close()
33
34 # To close AnaPico AG,APPH20G
35 AP.Close()
36
37 # To close 4-Channels Power Suppy GPP4323
38 GPP.Close()
```

In the flow chart 3.1 an example of the Automatic measurments python library construction is again given.

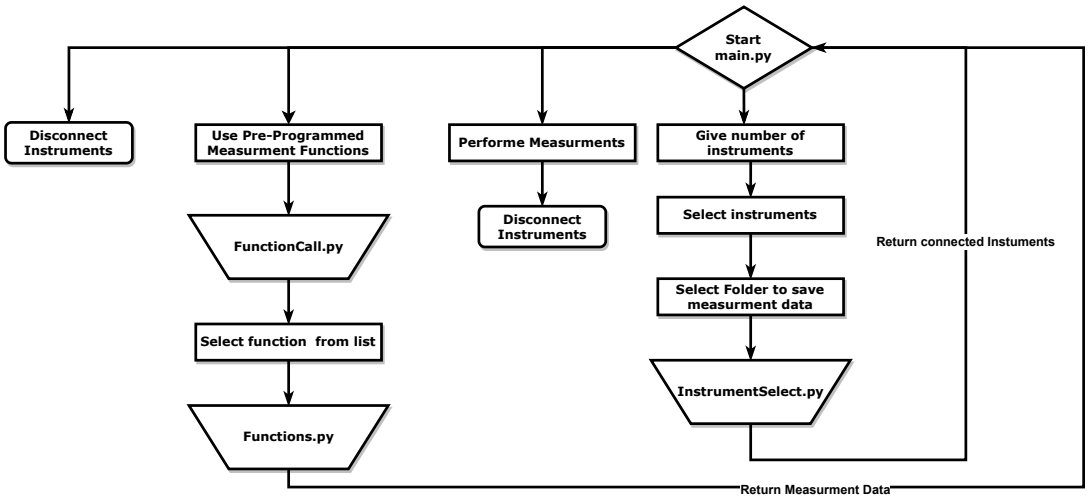



***Abbildung 3.1:*** *Flow Chart of the automatic measurments python scripts*

# 4 Example

In this chapter, we present an example along with a step-by-step picture tutorial on how to utilize the scripts. This example is conducted on the laboratory PC within the laser laboratory, where the Spyder API from the Anaconda distribution is employed.

Let's get started:

- **Step 1: Clone/Download the Automation Measurements Folder from Git**

Begin by cloning or downloading the 'Automation Measurements' folder from the Git repository. This folder contains all the necessary scripts, organized into different folders and sorted by name. In Figure 4.1, you can observe the files available in the 'Automation Measurements' folder, including the essential main.py file.

- **Step 2: Connect the Measurement Instruments and Verify Device Drivers**

Before you begin automated measurements, it's essential to ensure that your measurement instruments are properly connected, and the required device drivers are installed on your system. Follow these steps: 1) Connect your measurement instruments as per your specific setup requirements. 2) Verify that the necessary device drivers for your instruments are correctly installed and recognized by your system. You can check this in the 'Device Manager' on your computer.
(Note: Users should ensure that their instruments are correctly connected and drivers are installed. Specific instructions may vary based on the instruments used.)
In this example, we'll focus on using the Power Meter ThorLabs PM100D as the instrument. Later on, we'll apply the existing Coupling Stability function. The image in Figure 4.2 provides an example of a measurement setup to help you visualize the configuration. With your instruments properly connected and drivers confirmed, you're ready to proceed with the automated measurements

- **Step 3: Open the main.py Script**

To begin automated measurements, open the main.py script in your Python API environment. In this example, we are using Spyder. You can follow these steps:

1. Launch your Python API (Spyder) on your computer
2. Locate the main.py script within the 'Automation Measurements' folder that you previously cloned or downloaded

You can see part of the script's contents in Figure 4.3. Opening 'main.py' is the next step in setting up and running automated measurements. Let's proceed to configure and use the script.

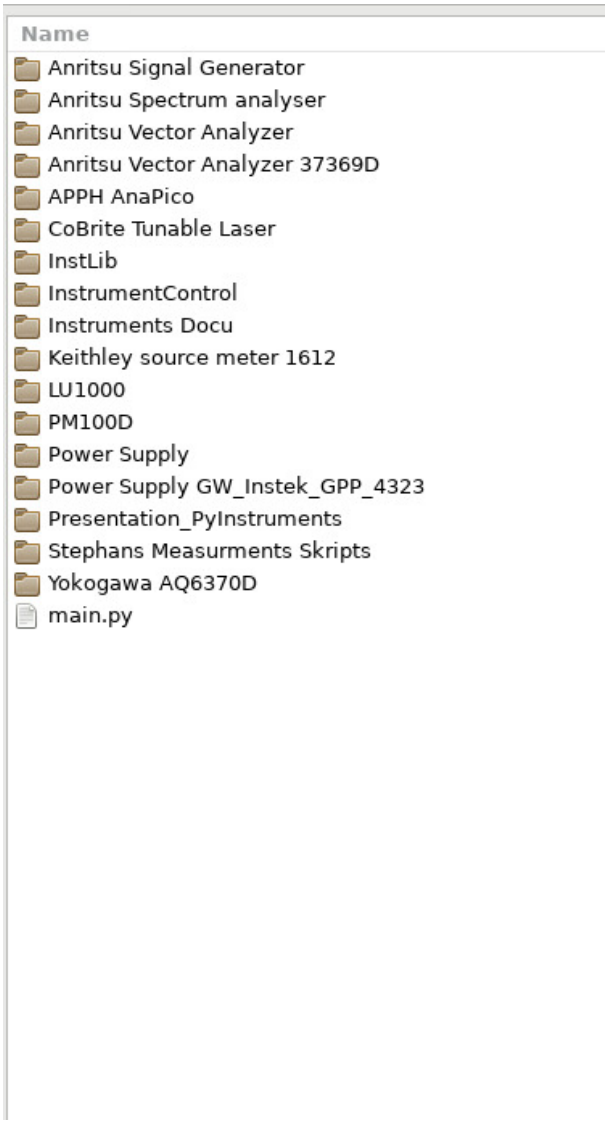



***Abbildung 4.1:*** *Folders in Auto_Measurement Git Directory*

- **Step 4: Start the Script**

With the main.py script open in your Python environment (e.g., Spyder), you're ready to initiate the script sequence by clicking the play button or running the script. Be sure to provide all the required information accurately as prompted. While we've incorporated safety error-catching functions into the scripts, not all scenarios have been thoroughly tested, as I am personally familiar with script operation. If you encounter any errors or can provoke errors that the scripts fail to catch, please report them promptly. Your feedback is invaluable for identifying and resolving any issues to enhance the reliability of the scripts.

- **Step 5: Input the Number of Instruments to Be Used**

Upon clicking the play button to start the script, the first prompt in the Python console


***Abbildung 4.2:*** *Measurement setup*

will ask you to specify how many instruments you intend to use. You will need to enter the correct number accordingly. Figure 4.4 provides a visual representation of what this prompt will look like in the console.

In this example, since i am using the Power Meter ThorLabs PM100D, i will select 1 for one instrument.

- **Step 6: Select the Instrument You Will Use**

After specifying the number of instruments you plan to use, the next prompt in the Python console will ask you to select the instrument from the list of instruments you intend to use. Simply enter the corresponding number for your instrument choice. Figure 4.5 provides a visual representation of what this selection prompt will look like in the console.

After selecting the instrument by entering the corresponding number, the script will proceed to automatically search all the USB and Ethernet ports on your PC for the connected instrument. Once a successful connection is established, you will receive a message indicating that the instrument is connected, such as: Power Meter ThorLabs PM100D is connected as PM. Please note the last initials, which, in this case, are PM. With these initials, you can now call functions from the Power Meter ThorLabs PM100D Library directly in the main.py file. For instance, if you wish to set the wavelength range on the device, you can easily do so by entering the following command in the Python console: PM.WaveLength(1550). This command will set the wavelength to (1550 nm). This

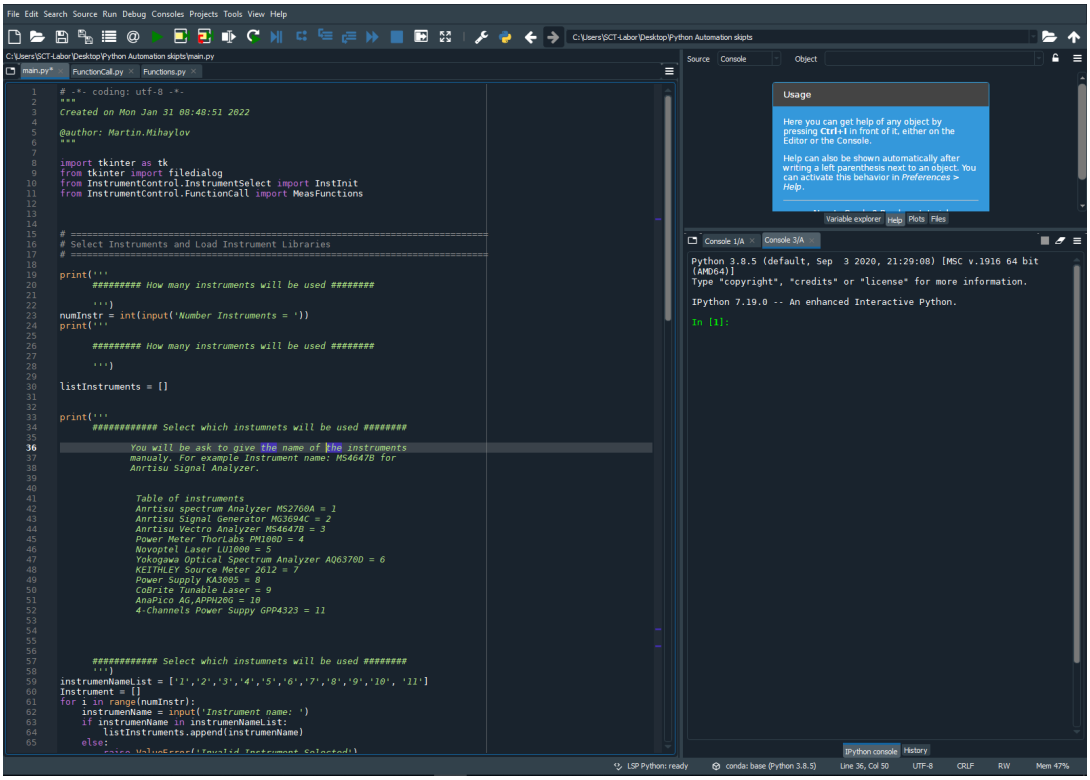



***Abbildung 4.3:*** *main.py opened in Spyder*

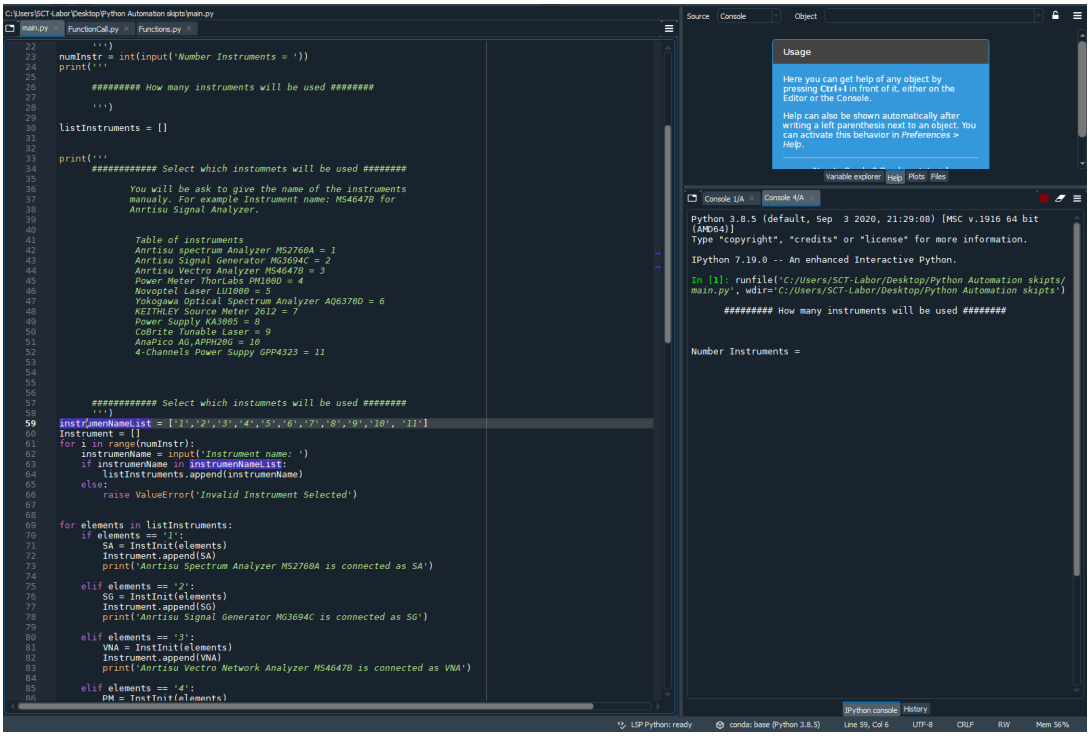



***Abbildung 4.4:*** *How many instruments will be used*

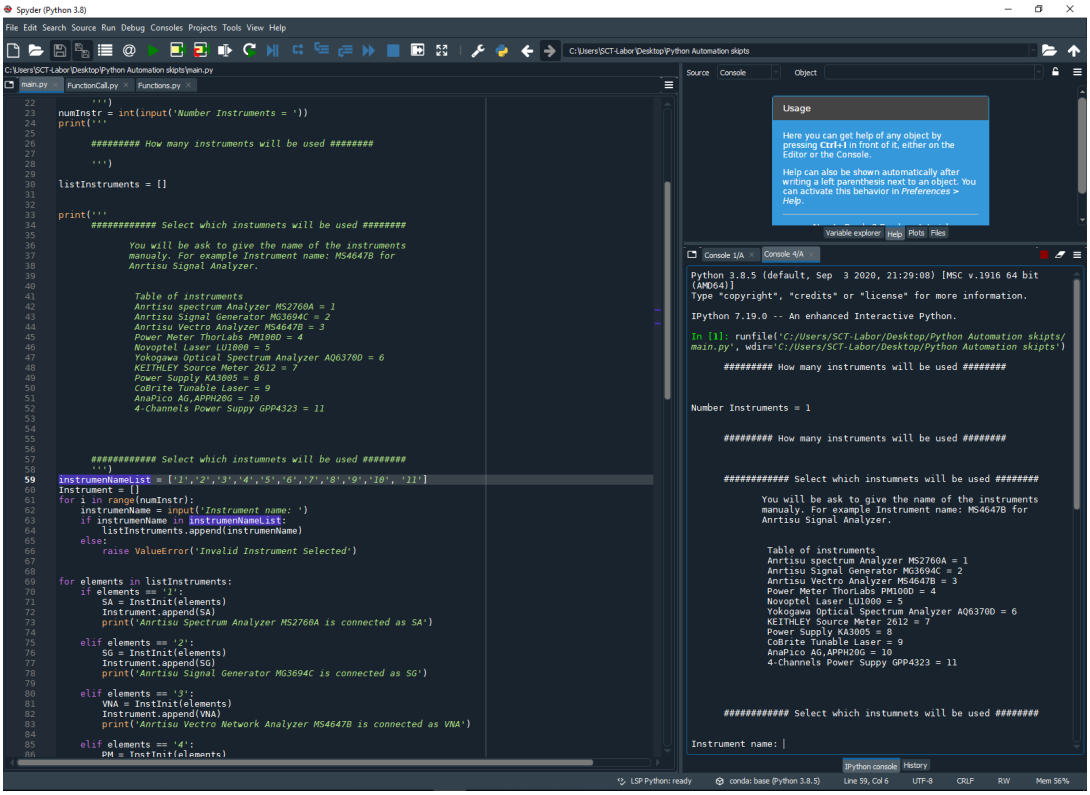


***Abbildung 4.5:*** *Which instruments will be used. Select form list*

step simplifies the process of connecting to instruments and accessing their functions, allowing you to control and configure instrument settings with ease.

- **Step 7: Choose the Location to Save Measurement Data**

After connecting to the instruments and configuring your measurements, the next important step is choosing where on your PC you want to save the extracted measurement information. As mentioned in Chapter 1, the script will generate three types of files for each measurement:

1. An .svg picture visualizing the measurement.
2. An .txt file containing all the parameters you set for the measurement.
3. A .csv file containing all the measured data, allowing you to plot the measurement graph as needed.

To facilitate this process, an automatic pop-up window (as shown in Figure 4.6) will appear, prompting you to navigate to the folder where you want to save this valuable data. This step ensures that your measurement data is efficiently organized and saved to the location of your choice for future analysis and reference.

You can now write your own measurment function or you can use one of the pre-programmed functions. To do so uncomment the following line in the main.py file.

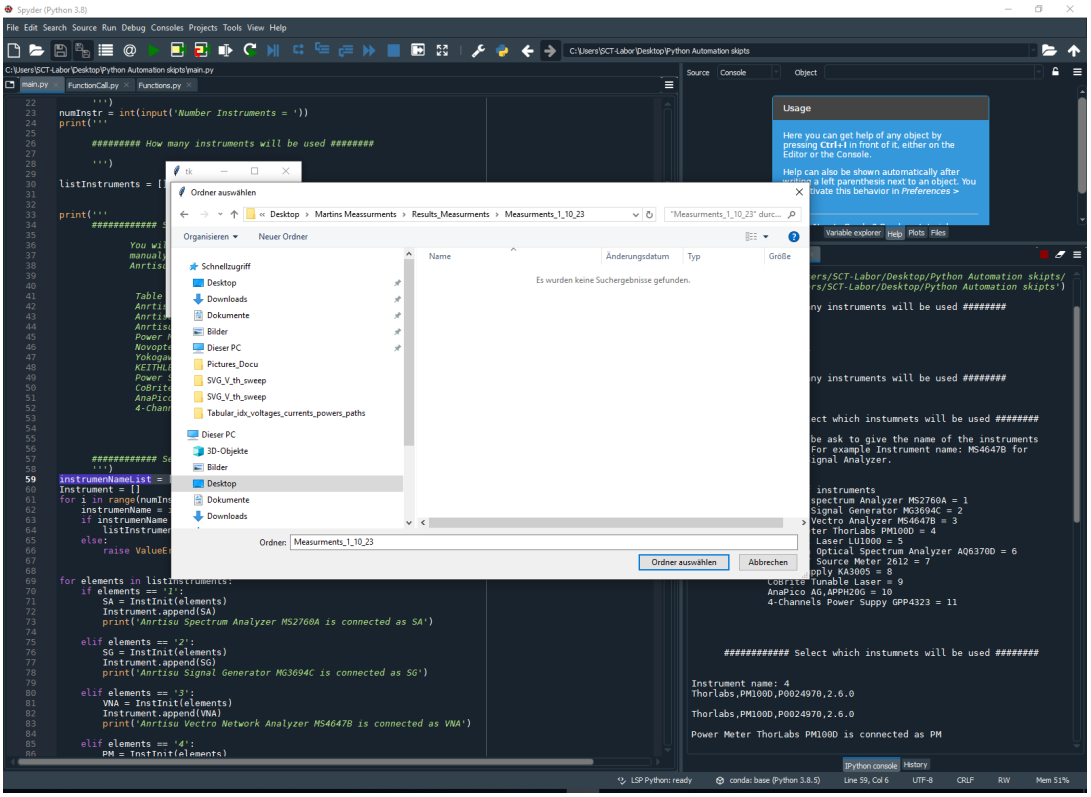



***Abbildung 4.6:*** *Select folder on your PC to save the data*

```
#Main Function. Instrument is a list of selected instruments for the
    measurment
MeasFunctions(Instrument,path)
```

- **Step 8: Use Pre-Programmed Measurement Functions (Optional)**

If you prefer to use the already pre-programmed measurement functions, you can do so with ease. Simply use the MeasFunctions(Instrument, path) function in the script. However, if you wish to skip this step and customize the script further, you have the flexibility to comment out this line based on your preference. For example, in my example scenario, I will use the MeasFunctions(Instrument, path) function to perform the existing 'coupling_stability' measurement. Upon using the MeasFunctions(Instrument, path), a new set of questions will appear in the Python console. The first question will prompt you to choose from one of the already existing functions, as shown in Figure 4.7.

- **Step 9: Set Measurement Data Parameters**

After selecting the pre-programmed measurement function, the script will prompt you to set all the necessary data parameters for the measurement. The required parameters for your chosen function will be clearly outlined in the docstring, which will be printed in the Python console. This docstring provides detailed guidance on what data needs to be configured for the measurement. For visual reference, Figure 4.8 illustrates how this

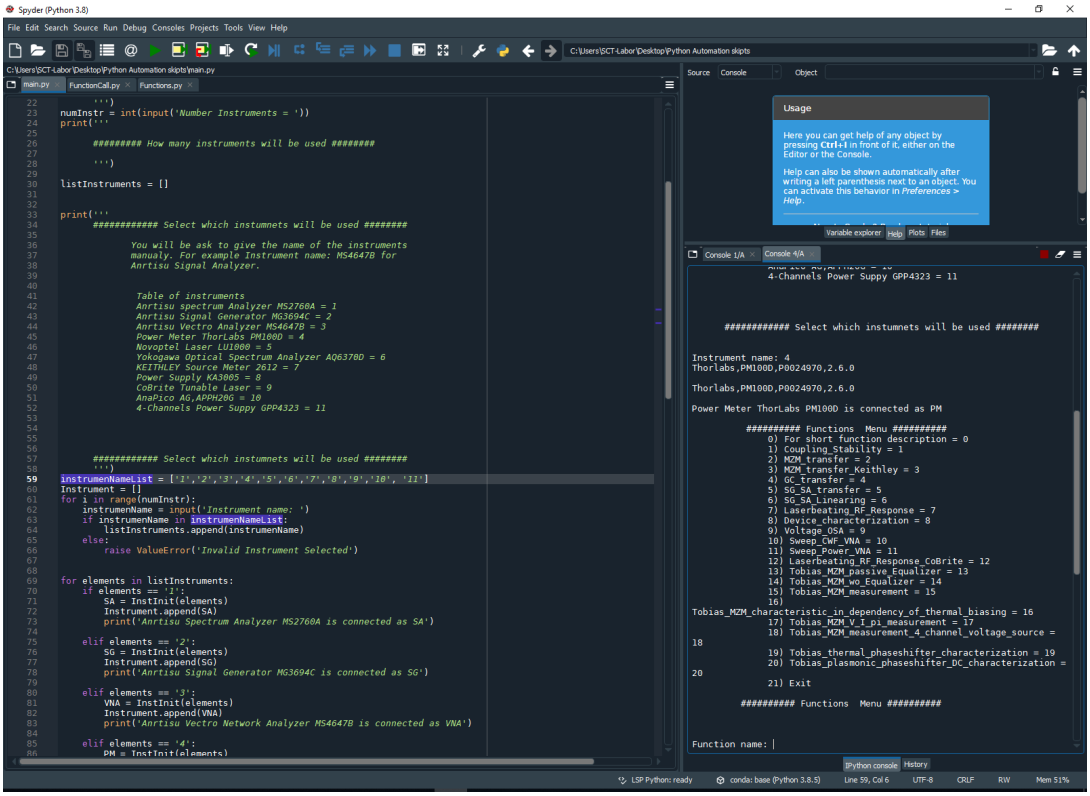


***Abbildung 4.7:*** *Select one of the pre-programmed measuring functions*

process appears in the Python console. This step ensures that you have full control over configuring the measurement according to your specific requirements, with the docstring serving as a comprehensive guide.

- **Step 10: Commence the Measurement**

Once you have provided all the necessary information and configured the measurement parameters, the measurement process will commence. In this example, it involves measuring the power at a given wavelength for a set duration. You will be able to observe the progress of the measurement in real-time. Figure 4.9 provides a visual representation of what the measurement process looks like in the Python console. You will notice a loading bar indicating the progress of the measurement, along with a countdown showing the number of sweeps completed and the remaining sweeps to be performed. This step marks the execution of the measurement, allowing you to track its progress and gather valuable data for analysis.

- **Step 11: Close Open Sockets and Disconnect Instruments**

Upon completing your measurements, it is crucial to close all open sockets. These sockets act as communication channels between your PC and the instruments. Failure to close these sockets can lead to issues, particularly with instruments like the VNA (Vector Network Analyzer), which may require a restart if the socket is not released. For the VNA,

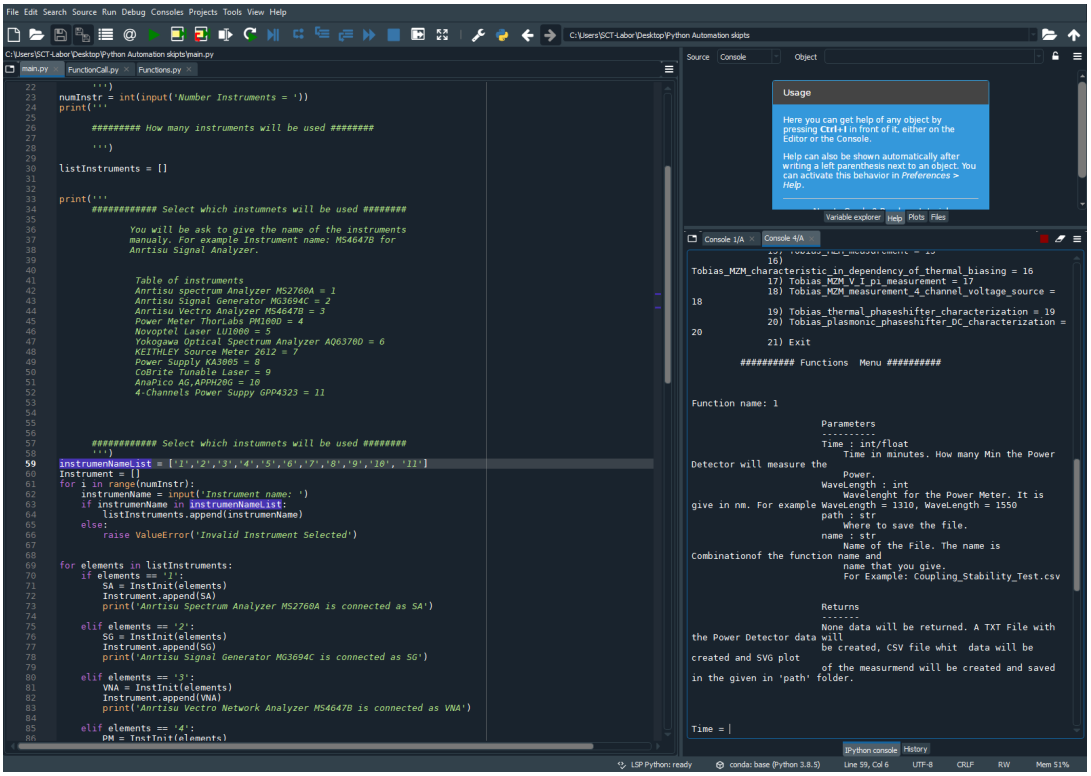


***Abbildung 4.8:*** *Set the required measurement parameters. In this section I am asked to set time, wavelength and name for my measurement files*

this means recalibrating the instrument, which can be time-consuming. To close the socket associated with a specific instrument, you can use a command in the following format: InstrumentName.Close(). Replace 'InstrumentName' with the initials provided by the Python console when connecting the instrument, followed by 'Close'. For example, to close the socket for the Power Meter ThorLabs PM100D, you can enter PM.Close(). Figure 4.10 provides a visual reference on how to close a socket and disconnect an instrument in the Python console. This step ensures that all communication channels are properly closed, preventing potential issues and ensuring a smooth workflow for future measurements.

Once your measurement is complete, you can conveniently access your measurement results in the folder you specified at the beginning of the process (see Figure 4.11 for reference). The data generated and saved includes:

1. An .svg file that visualizes the measurement

2. An .txt file containing all the parameters you configured for the measurement

3. A .csv file that holds all the measured data, enabling you to plot and analyze the results as needed (see Figure 4.12)

This step enables you to access and work with your measurement data for further analysis and interpretation.

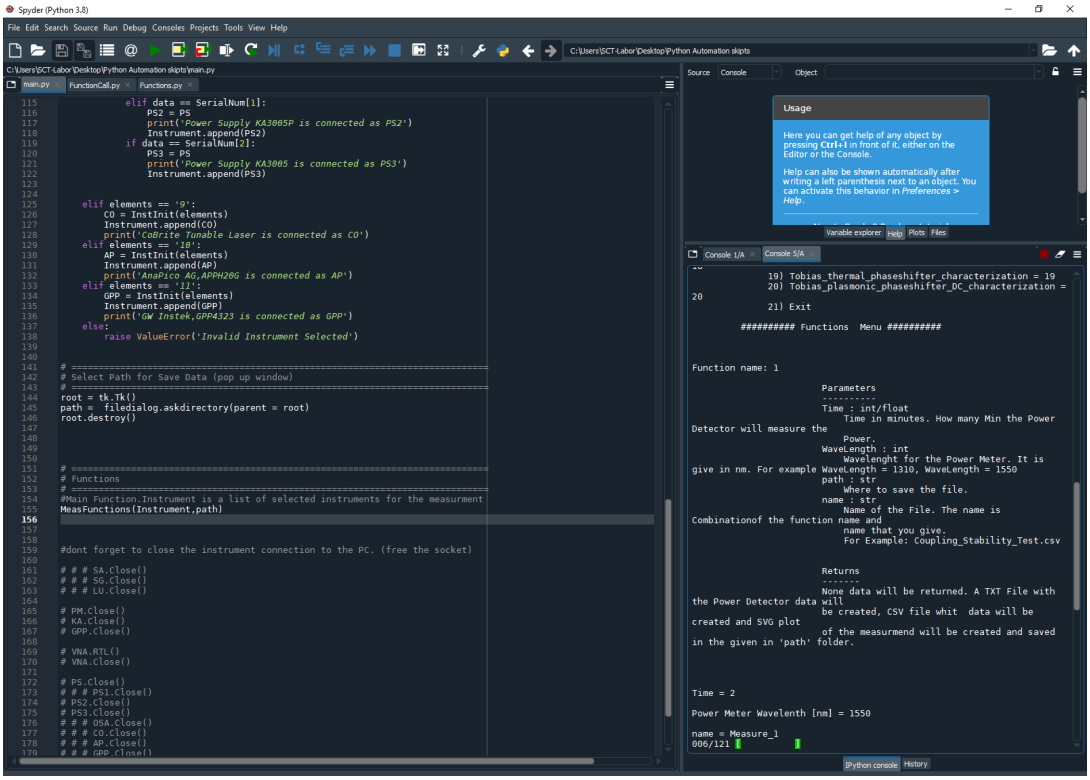


***Abbildung 4.9:*** *The measurement is set and running. The user can see how much is left and how many are done through the load bar function*

In this example, we've demonstrated the streamlined process of using the Automated Measurements Python script to perform measurements with ease. While we've used only one instrument for this illustration, it's important to note that the script supports multiple instruments, allowing you to harness the full range of capabilities if needed. Moreover, we've showcased the convenience of pre-programmed measurement functions. These functions are ready to use for quick measurements. However, if you have specific measurement needs or wish to create custom functions, you have the flexibility to reach out for assistance, or if you're proficient in Python, you can build custom functions yourself. The Automated Measurements Python script empowers you to efficiently control and analyze measurements, making it a valuable tool in various laboratory and research settings. Feel free to explore, adapt, and customize the script to suit your unique requirements. Your measurement capabilities are now in your hands, whether you need swift measurements or tailor-made solutions.
Should you have any questions or require assistance with custom functions, don't hesitate to reach out. Happy measuring!

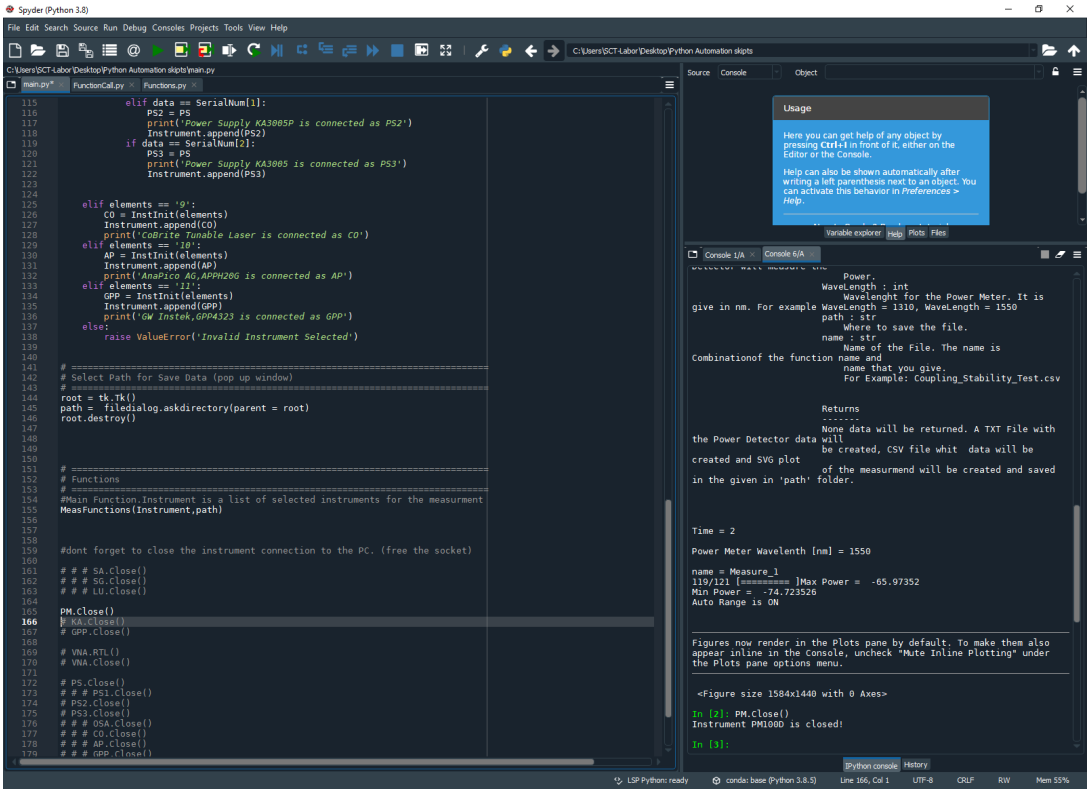


***Abbildung 4.10:*** *Disconnect the device from the mains after use*

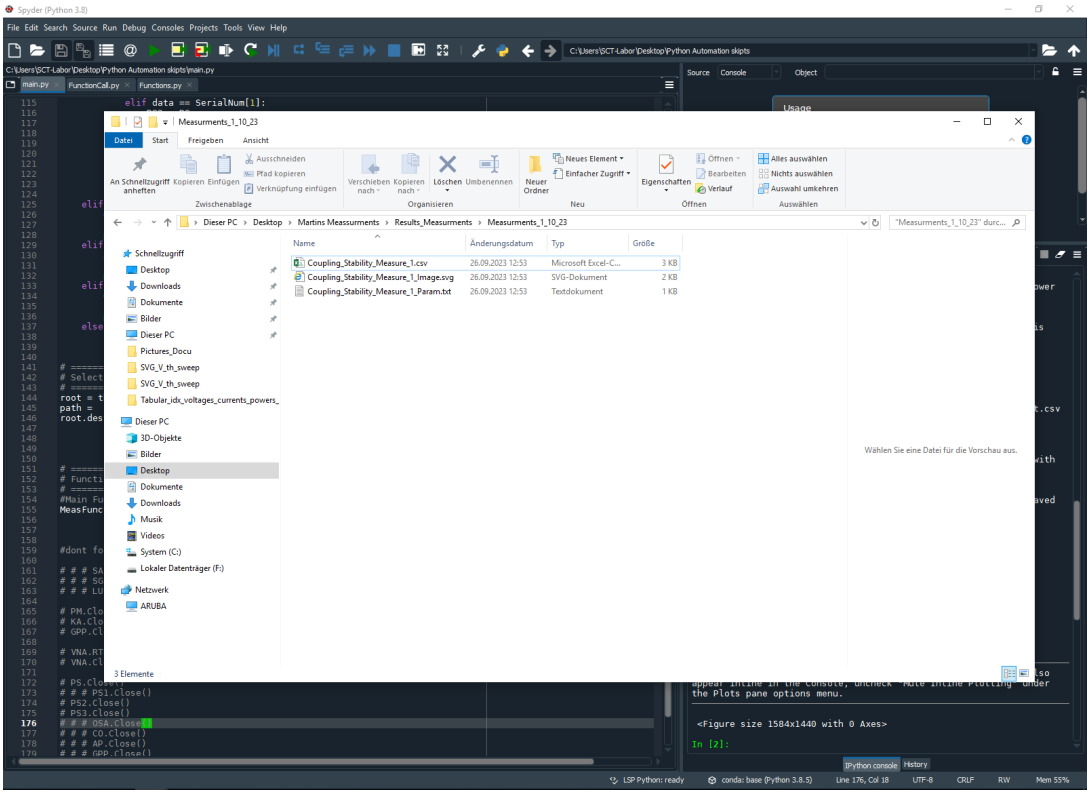


***Abbildung 4.11:*** *Data saved on PC/Laptop after measurement*

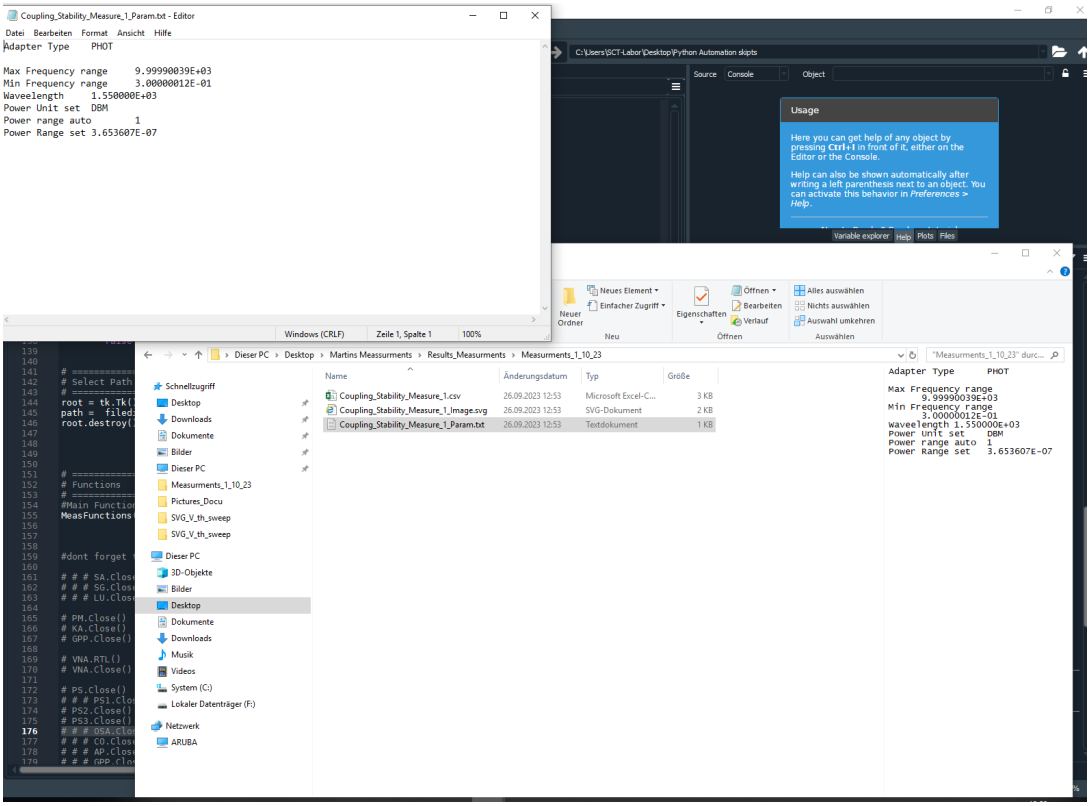



***Abbildung 4.12:*** *Contents of the .csv file*