



Hochschule
für angewandte Wissenschaften
Würzburg-Schweinfurt

GE-Lab course

Generation of tones with python and a waveform generator

Laboratory for communication technology

Generation of tones with Python and a waveform generator

The Goal of this Experiment is to get a communication between Python and the Keysight 33510 Waveform Generator, to produce various functions, that can be played on headphones and displayed on an oscilloscope. In order to reach this goal, the students will get some basic knowledge about programming with Python.

About Python

Python is a Script based programming language that was developed with the goal to keep it simple and clear. There are many purposes one can use Python, either for structured, or functional or even for object-oriented programming.

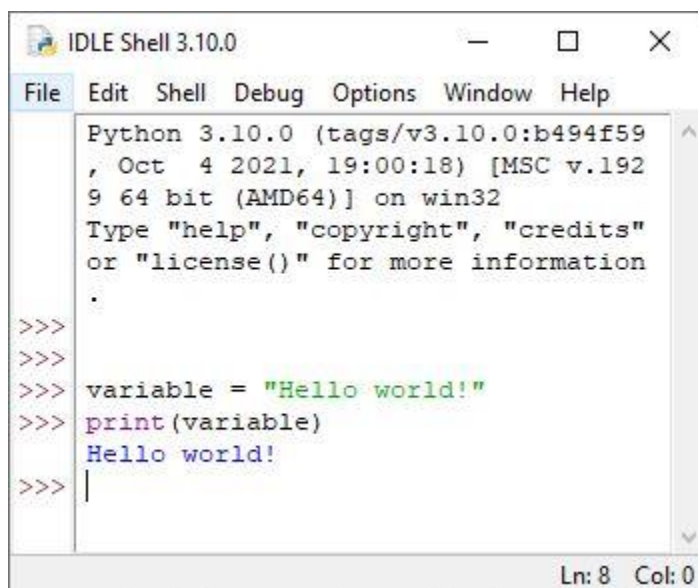
Python is a very mighty tool for a fast implementation of Applications, based on its huge variety of offered libraries, like the one that will be used here (pyvisa).

Getting Started

Here the used "editor" is the Python IDLE Shell. It's like a command line and very useful, if you want to try out some things fast, but it's not so useful for scripting, because you can't correct written text, instead you must rewrite a whole block, even if it's only a typo. Recommended editors for this Experiment are PyCharm, Thonny, Atom, Spyder, or you could use yours if you already installed one.

To start, there are some basics you must know about Python. At first the print-Statement. It prints the content that's given to it in the console. It can print Strings or the content of various variables. Also, it's important to know, that comments are marked with # for lines and ''' for Block comments.

E.g.:

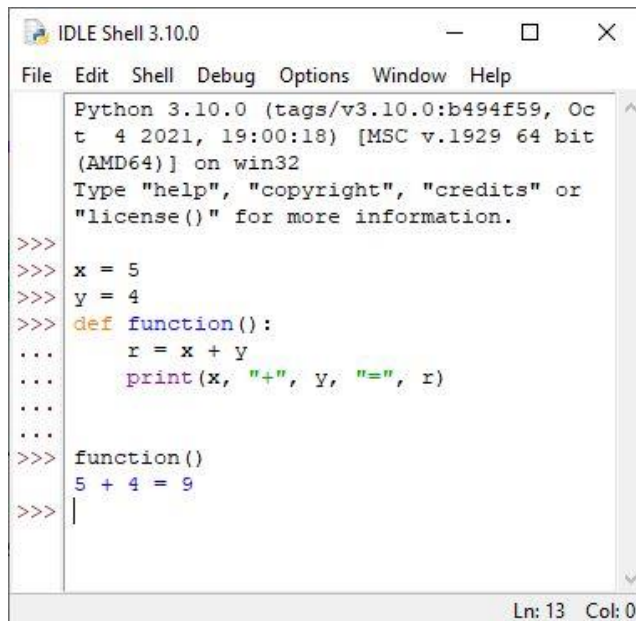


```
Python 3.10.0 (tags/v3.10.0:b494f59
, Oct 4 2021, 19:00:18) [MSC v.192
9 64 bit (AMD64)] on win32
Type "help", "copyright", "credits"
or "license()" for more information
.>>>
>>>
>>> variable = "Hello world!"
>>> print(variable)
Hello world!
>>> |
```

Figure 1: Print example

To implement functions Python differences blocks via the white space.

E.g.:

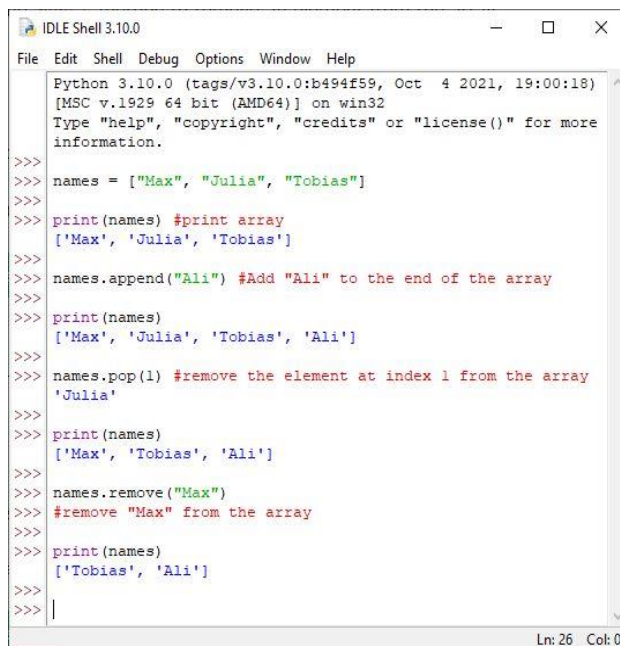


```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> x = 5
>>> y = 4
>>> def function():
...     r = x + y
...     print(x, "+", y, "=", r)
...
>>> function()
5 + 4 = 9
>>> |
```

Figure 2: Function example

It's also important to know something about arrays in Python. An array is a variable, that can hold more than one item at once. E.g., if you have a list of names you want to save, you can use an array instead of many single variables. An array in Python is, unlike in C or C++, extendable or items can also be removed. If you want to print your array, you simply can use the print function.

E.g.:

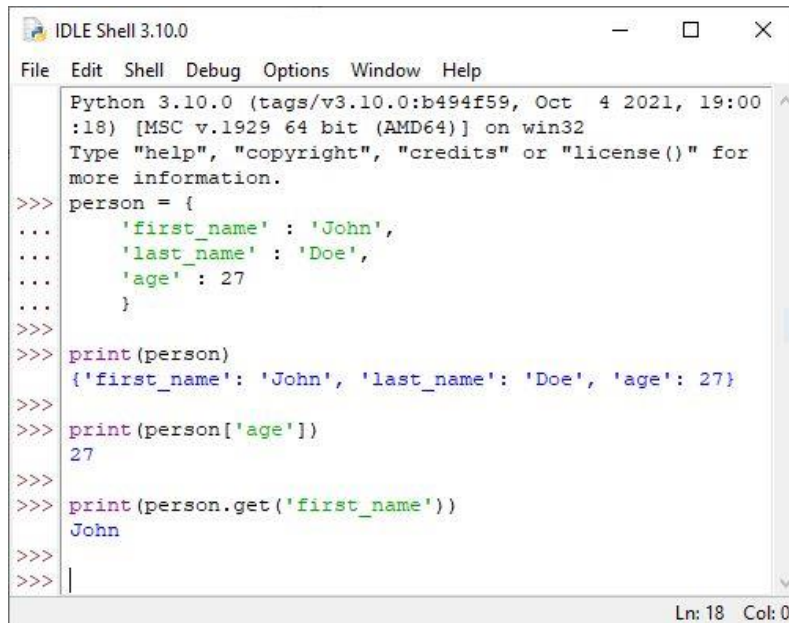


```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> names = ["Max", "Julia", "Tobias"]
>>>
>>> print(names) #print array
['Max', 'Julia', 'Tobias']
>>>
>>> names.append("Ali") #Add "Ali" to the end of the array
>>>
>>> print(names)
['Max', 'Julia', 'Tobias', 'Ali']
>>>
>>> names.pop(1) #remove the element at index 1 from the array
'Julia'
>>>
>>> print(names)
['Max', 'Tobias', 'Ali']
>>>
>>> names.remove("Max")
>>> #remove "Max" from the array
>>>
>>> print(names)
['Tobias', 'Ali']
>>>
>>> |
```

Figure 3: Array example

Similar to an array is the dictionary. It can connect values with keywords, for example the name, age and address from a person, or in the case used here, you can specify different frequencies for different notes.

E.g.:



```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> person = {
...     'first_name' : 'John',
...     'last_name' : 'Doe',
...     'age' : 27
... }
>>>
>>> print(person)
{'first_name': 'John', 'last_name': 'Doe', 'age': 27}
>>> print(person['age'])
27
>>> print(person.get('first_name'))
John
>>>
>>> |
```

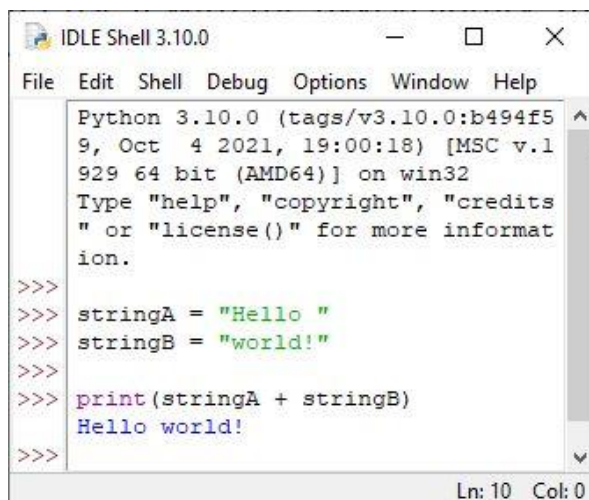
As you can see, there are different types of values in that dictionary. It is possible to write every type of value to a keyword. E.g., you could also set a keyword 'names' and write a list with all names of that person to it.

If you'd try to access a keyword, that doesn't exist, it will throw a `KeyError`.

For this experiment it's useful to write a dictionary, that provides the frequencies of different notes as a String, to later use it with the `pyvisa` library, that communicates with the waveform generator. You will find a small list with tones and frequencies on page 5.

Hint: In Python it's easy to connect different Strings by using the add-operation.

E.g.:



```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> stringA = "Hello "
>>> stringB = "world!"
>>>
>>> print(stringA + stringB)
Hello world!
>>>
```

Preparation

This Experiment attend to create different waveforms and signals on a signal generator with the use of Python. The used signal generator is a "KEYSIGHT 335010B Waveform Generator". It comes with several functionalities. With this waveform generator it's possible to create several waveforms:

Sine, Square, Ramp, Pulse, and Triangle. There is also the possibility to create Noise, PRBS (Pseudo random binary signal) and a DC signal.

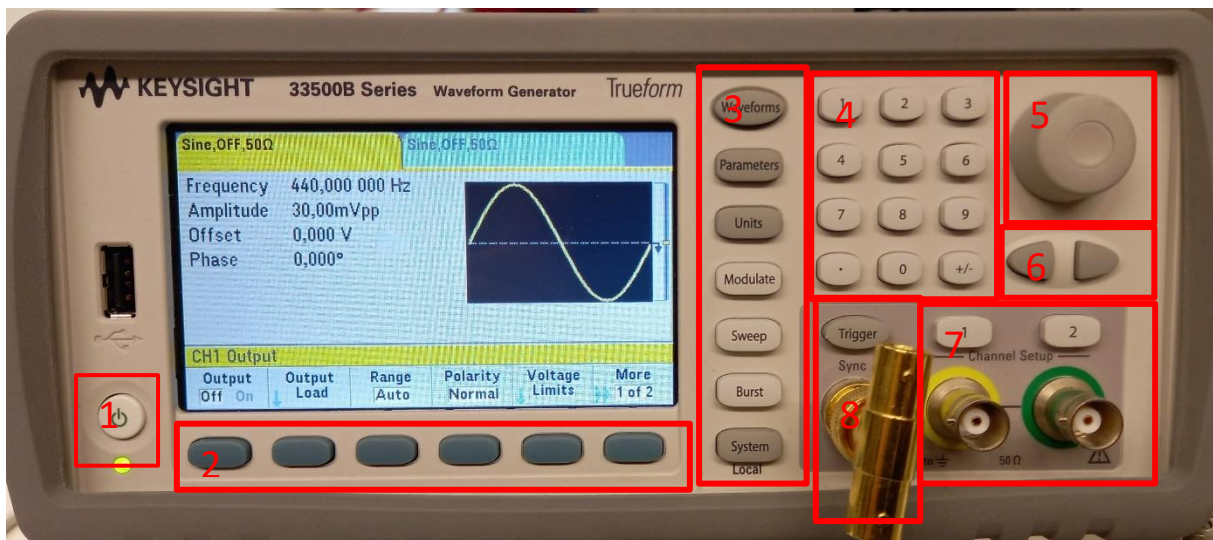


Figure 4: Waveform Generator front

- 1: ON/OFF
- 2: Selection of the options on the Display
- 3: Waveform selection and options
- 4: Numeric field to change parameters
- 5: Knob to change parameters
- 6: Selection of digit
- 7: Channel 1 (hereinafter: Ch1) and Channel 2 (hereinafter: Ch2) output
- 8: Trigger input

If you use the Generator without PC:

To choose a waveform, you must click on the Waveforms-button (3), then you can choose with the Selection buttons (2), which waveform you want. To change the parameters of the function, you click on "Parameters" (3) and then you can use the Selection buttons (2) to choose the parameter you want to change. With the Numeric field (4) or the Knob (5) you now can change the value of the parameter.

To turn on or off a channel you click on the button above the output of Ch1 or Ch2 (7) and then select Output on the selection buttons (2).

The Oscilloscope that will be used is a “Keysight InfiniiVision DSOX2024A” digital oscilloscope with several useful functions. It has 4 channel inputs, but we will only use 2 of them.

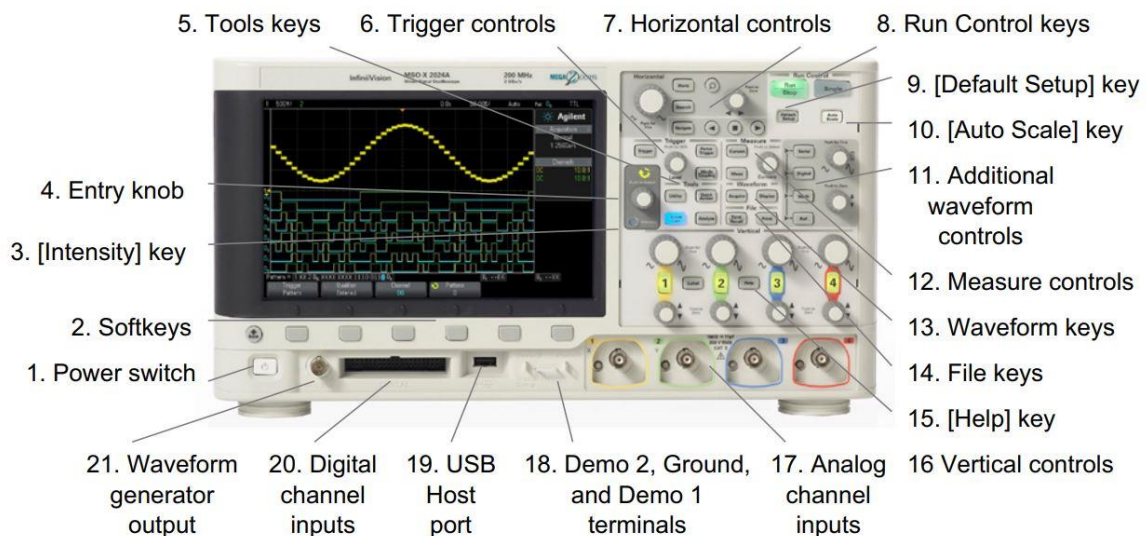


Figure 5: Oscilloscope functions

To scale the input signal, you can use the Auto Scale key (10), or you do it manually by using the Horizontal and Vertical controls (7, 16).

With the Measure controls (12) you can take measurements on the display of the oscilloscope. To take a measurement, you must click on “Cursors”, then use the “Cursor” knob to move and select the cursor.

With the Softkeys (2) you have the possibility to choose between several options shown on the bottom of the display.

A very interesting thing is the opportunity to analyze an input signal, e.g., it’s possible to do a Fourier transformation with the oscilloscope on an input waveform.

Now: Some music theory

On principle are tones sinewaves that are overlapped. A well-known tone for musicians is the “Concert pitch” or also named A4 (English notation)/ a¹ (German notation). It has a frequency of 440 Hz and is used to tune instruments.

For the following experiment you need to know this tone, but also a few other tones and their frequencies. The following table gives a short overview about them.

This is one scale (from C4 to C5) with all the tones in between. Now, some music theory. If you want to know the frequency of a note that isn’t in that table, you only need to multiply or divide the value of the octave note’s frequency.

E.g., the frequency of E5 is the frequency of E4 doubled ($2 \cdot 329.6 \text{ Hz} = 659.2 \text{ Hz}$). For D3 it would be $293.7 \text{ Hz} / 2 = 146.85 \text{ Hz}$.

Now, if you want to play some music from your waveform generator, you must know something about notation and the speed how you want to play music.

Basically, notes aren't written as text, this would be a mess, because a note hasn't always the same length. E.g., there are half notes or quarter notes etc. The main theory is pictured in the following.

The diagram illustrates various musical notation concepts using a 4/4 time signature:

- Treble clef:** Indicated by a red box around the clef.
- Eighth notes:** A sequence of eight eighth notes is shown, with a red box around the first two labeled "beat".
- bar:** A vertical line separating measures, shown as a purple box around a bar line.
- Full note:** A single whole note (semibreve) is shown.
- Half notes:** Two half notes (minims) are shown.
- Eighth notes with accidentals:** A sequence of eighth notes with various accidentals (sharp, flat, natural) is shown, with boxes around the notes.
- Breaks:** Four types of rests are shown: Full break (whole rest), Half break (half rest), Quarter break (quarter rest), and Eighth breaks (eighth rest).
- Quarter notes:** A sequence of four quarter notes (crotchets) is shown.

The **Treble clef** intends the used notation. It says the second last line is the tone G4 and every line up, the next upper line would be the H4, and the next last line would be the E4. Each line to an interspace up or down is a full tone. And, from every interspace to a line up or down. E.g., the room above the G4-line is the tone A4, the "Concert pitch".

The **beat** gives how many notes of which note type are played in one full beat. E.g., the given four-quarter beat says one beat must consist of a combination of notes that fill four quarters, like a full note, two half notes and so on. One beat is limited by a **bar**. Often a speed (in beats per minute) is given at the beginning of a piece, if so, you need to calculate the duration of one quarter note. If no speed is given, assume it is 100 bpm (beats per minute). Bpm gives how many quarter notes are played in one minute.

There are also three common **accidentals**, the **sharp**, the **flat** and the **natural**.

The **sharp** raises the pitch of a note one semitone, the **flat** lowers a note one semitone. The **natural** cancels the effect of a sharp or a flat. But not every note has a decreased or raised neighbor semitone. E.g., there is no E# or no Fb!

E.g.:

The diagram illustrates the effect of accidentals on a note:

- Raises C5 to C#5:** A sharp symbol (#) is shown above a C note on the fifth line.
- Low E4 to Eb4:** A flat symbol (b) is shown below an E note on the fourth line.
- Cancels effects, so C#5 is now again C5 and Eb4 is now again E4:** A natural symbol (♮) is shown below a C# note on the fifth line and a sharp symbol (#) is shown above an Eb note on the fourth line.
- This would last the whole piece:** A key signature change is shown, with a flat symbol (b) on the F line and a sharp symbol (#) on the C line.

A accidental noted in one beat lasts only as long as the beat or as long as it's not canceled.
Accidentals could also operate in the key signature, so their effect would continue the whole piece.

The last thing you may need is a full scale:



Now you should know enough about music theory to implement a little song in python, that should be generated by the waveform generator.

The PyVISA-Library

The PyVISA-Library is especially for communication with laboratory-equipment, like function generators, oscilloscopes or other measurement systems. It provides communication for Systems from different manufacturers, like Keysight, Tektronix, R&S and NI, which all use the IVI-library that is included in PyVISA.

WARNING: PyVISA works with 32- and 64-bit Python and can deal 32- or 64-bit VISA libraries without extra configuration. But it can't open a 32-bit VISA library while running in 64-bit Python or the other way around!

How to install PyVISA

You can install PyVISA using the pip command. This may not work on a Windows computer, so you have to install pip first. But this depends either on your used IDE or on your installed Python Version.

If you have installed the latest Version from the official site, you should not get any problems. If you use another IDE, like PyCharm or Thonny, etc. you should type this command in the terminal of your IDE, but if you use IDLE Python from the Python package you can type it into your command window (cmd).

```

C:\Users\tosch>pip install -U pyvisa
Collecting pyvisa
  Using cached PyVISA-1.11.3-py3-none-any.whl (189 kB)
Requirement already satisfied: typing-extensions in d:\programs\python\python310\lib\site-packages (from pyvisa) (4.0.1)
Installing collected packages: pyvisa
Successfully installed pyvisa-1.11.3
WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.
You should consider upgrading via the 'D:\Programs\Python\Python310\python.exe -m pip install --upgrade pip' command.

C:\Users\tosch>
  
```

Figure 6: install pyvisa

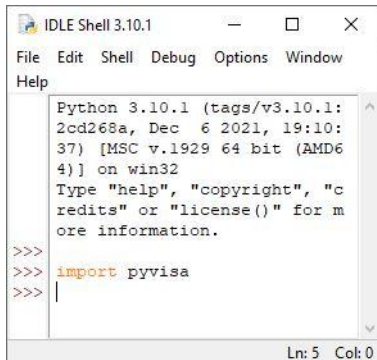
If it doesn't work, try to manually install pip by using following command (same for windows, linux and mac):

```
python -m get-pip.py
```

And now try again installing PyVISA

Using PyVISA

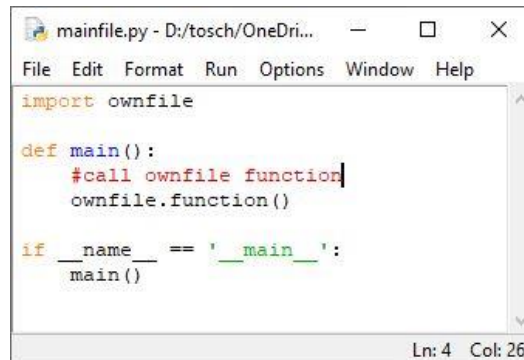
First you need to import the pyvisa library:



```
Python 3.10.1 (tags/v3.10.1:
2cd268a, Dec 6 2021, 19:10:
37) [MSC v.1929 64 bit (AMD6
4)] on win32
Type "help", "copyright", "c
redits" or "license()" for m
ore information.
>>>
>>> import pyvisa
>>>
```

Figure 8: import pyvisa

You can import own files the same way:



```
import ownfile

def main():
    #call ownfile function
    ownfile.function()

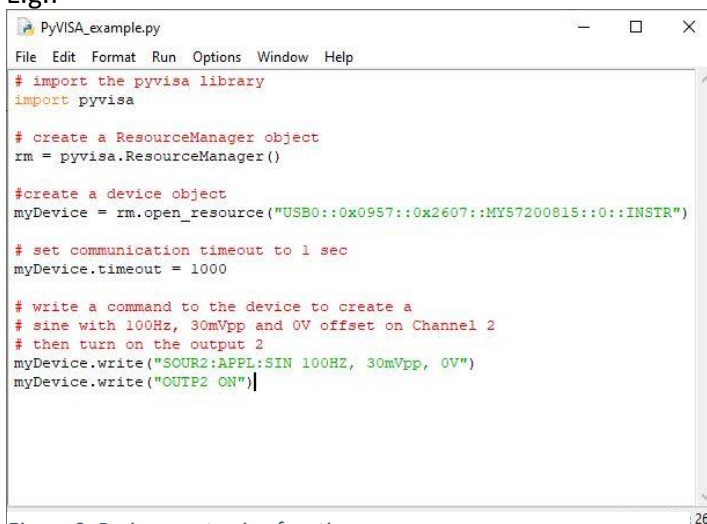
if __name__ == '__main__':
    main()
```

Figure 7: import own file

Using the PyVISA library is very simple, you basically just need 5 commands.

1. [ResourceManager]= ResourceManager() – It creates an object, that now can open the communication port
2. [Resource]= [ResourceManager].open_resource(“Resource Address”¹)
3. [Resource].timeout – sets a defined communication timeout for your device
4. [Resource].write(“Command”) – sends a command (String) to your device
5. [Resource].read() – reads an answer from the device

E.g.:



```
PyVISA_example.py
File Edit Format Run Options Window Help

# import the pyvisa library
import pyvisa

# create a ResourceManager object
rm = pyvisa.ResourceManager()

# create a device object
myDevice = rm.open_resource("USB0::0x0957::0x2607::MY57200815::0::INSTR")

# set communication timeout to 1 sec
myDevice.timeout = 1000

# write a command to the device to create a
# sine with 100Hz, 30mVpp and 0V offset on Channel 2
# then turn on the output 2
myDevice.write("SOUR2:APPL:SIN 100HZ, 30mVpp, 0V")
myDevice.write("OUTP2 ON")
```

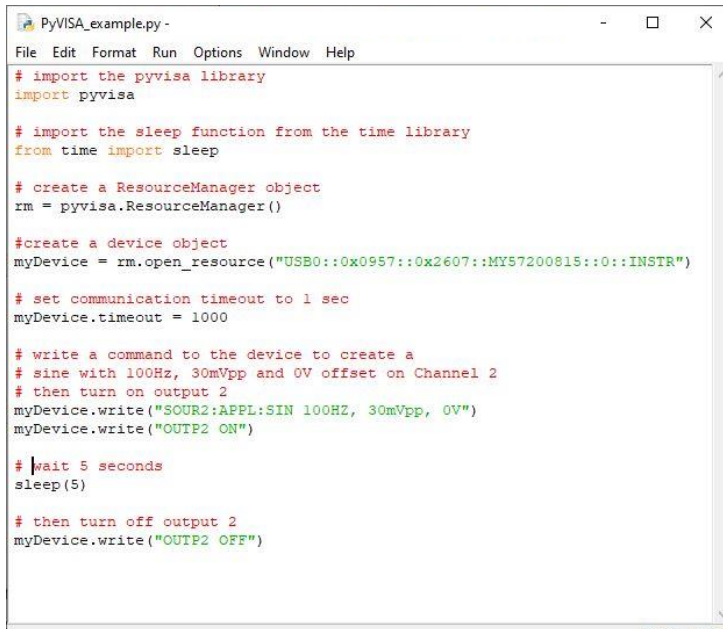
Figure 9: Pyvisa create sine function

This example creates a sinewave with a frequency of 100Hz, a Magnitude of 30mVpp (mV peak-peak) and 0V offset.

¹ To find out the resource address you need to launch the Keysight Connection Expert. How this works is described in the document “How to install and use Keysight Connection Expert”.

The second library you need is the time-library, especially the sleep function. It is needed to set a delay between turning on and off the outputs of the waveform generator. The sleep function works with seconds.

E.g.:



```
PyVISA_example.py
File Edit Format Run Options Window Help

# import the pyvisa library
import pyvisa

# import the sleep function from the time library
from time import sleep

# create a ResourceManager object
rm = pyvisa.ResourceManager()

# create a device object
myDevice = rm.open_resource("USB0::0x0957::0x2607::MY57200815::0::INSTR")

# set communication timeout to 1 sec
myDevice.timeout = 1000

# write a command to the device to create a
# sine with 100Hz, 30mVpp and 0V offset on Channel 2
# then turn on output 2
myDevice.write("SOUR2:APPL:SIN 100HZ, 30mVpp, 0V")
myDevice.write("OUTP2 ON")

# wait 5 seconds
sleep(5)

# then turn off output 2
myDevice.write("OUTP2 OFF")
```

It's the same example as one page before, but now the output is turned off again after 5 seconds.

Figure 10: Pyvisa and time create sine function, turn it on and off

The following list sums up a few commands for the waveform generator you need to know. Every command you send from Python needs to be a String format!

*CLS – Clear event status registers

*IDN? – Device identification

SOUR[x]:APPL:[function] [frequency], [amplitude], [offset]

[function]: SIN(usoid)/TRI(angle)/SQU(are)/PULS(e)/RAMP/NOIS(e)/PRBS²

[frequency]: value unit e.g., 100 HZ / 1 kHz

[amplitude]: value unit e.g., 30 mVpp

[offset]: value unit e.g., 1 V

[x]: channel number (1/2)

The PULSe and the RAMP functions have parameters, that can be defined by special commands:

SOUR[x]:FUNC:PULS:DCYC [percent] – sets the duty cycle for the pulse function

(can also be used for SQUare function)

e.g. SOUR1:FUNC:PULS:DCYC 50 – sets the duty cycle at channel 1 to 50%

² PRBS - Pseudo random binary signal

SOUR[x]:FUNC:PULS:WIDT [value] – sets the pulse width (! Pulse width < period – w_{\min} !)

e.g. SOUR1:FUNC:PULS:WIDT 5ms – sets the pulse width at channel 1 to 5ms

SOUR[x]:TRAC [ON/OFF/INV] – Channel 1 and 2 output the same or an inverted polarity signal

e.g., SOUR2:TRAC:INV – Channel 2 outputs inverted signal from channel 1

SOUR[x]:PHAS [φ] – Sets the phase of a function – use with UNIT:ANGL:DEG, set the angle to degree

e.g. UNIT:ANGL:DEG

SOUR1:PHAS 90

➔ Now channel 1 has a phase of 90°

OUTP[x] [ON/OFF] – turns on or off the output x

IMPORTANT: The output load on the waveform generator must be set to 33Ω! This can be done by using following command: OUTP[1/2]:LOAD 33

Tasks:

1. Please Prepare a Table at home with all the frequencies from C3 to C5!
2. Why is it important to set the output load to 33Ω?
3. Write a Python script that identifies the connected device.
4. Write a script that generates the music scale from page 7 once up and down on the waveform generator. Do this follow these steps (please prepare steps a) – c) at home):
 - a. Write the dictionary with all notes and frequencies in an extra file named notedictionary.py
 - b. Write a function dictionary, that refers a pyvisa function to a keyword (e.g., “sine”: “SOUR[x]:APPL:SIN”). Save it as functiondictionary.py
 - c. Write a function play Note, that uses the dictionaries to play a note for a specific time. The function should use the following parameters: “note”, “noteDuration”, “pieceSpeed” and “function”. Think about how to calculate the duration of a note with the given speed and the given note duration. Save it to playfunction.py.
 - d. Now write a main function to call the playNote function with the correct parameter values to play a scale. Write a file named scale.py, that is called in main.
5. Now vary the function of the played note and describe the differences you can hear. Explain why there are differences between the various functions.
6. Now play some pieces of music:
 - a. Bring your national anthem and play it. Write a file named anthem.py
 - b. Use one of the following pieces (Write a file named piece.py)

1)



Hint: a point behind a note means it is 1.5 times long as its normal value.

2)

Ein Pro - sit, ein Pro - sit der Ge -

müt - lich - keit, ein Pro - sit, ein

Pro - sit der Ge - müt - lich -

keit! Schenkt ein, trinkt

aus, schenkt ein, trinkt aus! _____

How your files should look like:

Your scale.py, anthem.py and piece.py should look like this, just with the values of the notes you want to play:



```

*mainexample.py - D:/tosch/OneDrive/Dokumente/Studium...
File Edit Format Run Options Window Help
import playFunction as pf #include your playFunction

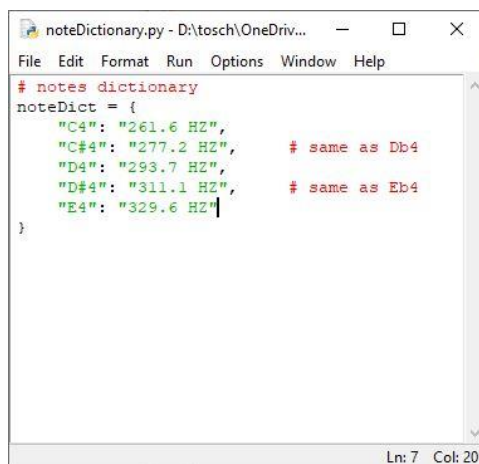
def main():
    pf.playNote("C4", "quarter", 100, "triangle");
    pf.playNote("D4", "quarter", 100, "sine");
    pf.playNote("E4", "quarter", 100, "square");
    # In general the function call should look like this:
    # pf.playNote(NOTE, DURATION, SPEED, FUNCTION);
    # Now it's all your turn!

# call the main function:
if __name__ == '__main__':
    main()
Ln: 1 Col: 54

```

Figure 11: main function example

noteDictionary.py:



```

noteDictionary.py - D:/tosch/OneDrive...
File Edit Format Run Options Window Help
# notes dictionary
noteDict = {
    "C4": "261.6 HZ",
    "C#4": "277.2 HZ", # same as Db4
    "D4": "293.7 HZ",
    "D#4": "311.1 HZ", # same as Eb4
    "E4": "329.6 HZ"
}
Ln: 7 Col: 20

```

Figure 12: noteDictionary example

functionDictionary.py:




```

functionDictionary.py - D:/tosch/...
File Edit Format Run Options Window Help
funcDict = {
    "sine": "SOUR1:APPL:SIN ",
    "triangle": "SOUR1:APPL:TRI "
}
Ln: 3 Col: 37

```

Figure 13: functionDictionary example

playFunction.py (### means it's a part you must do on your own):



```

playFunction.py - D:/tosch/OneDrive/Dokumente/Studium/HiWi/Lösungen/playFunction.py (3.10.1)
File Edit Format Run Options Window Help

# include your files
import functionDictionary as fd      #include your functionDictionary file
import noteDictionary as nd         #include your noteDictionary file

# now include system libraries
import pyvisa                      #include the pyvisa library
from time import sleep             #the sleep function is needed for the timing

# before you can play any note, you need to connect your device:
rm = ###
device = ###

# set your devices timeout to lsec:
device.timeout = 1000

# playNote function
# parameters: note, noteDuration, pieceSpeed (default 100), function
def playNote(note, noteDuration, pieceSpeed=100, function="sine"):
    # you can use the following dictionary to calculate the duration of a note
    durationDictionary = {
        "full" : ###,
        "half" : ###,
        "half_pointed" : ###,
        "quarter" : ###,
        "quarter_pointed" : ###,
        "eighth" : ###,
        "eighth_pointed" : ###,
        "sixteenth" : ###
    }

    # to play a note you now can call the functionDictionary
    # and the noteDictionary from your files
    # you only have to add the Magnitude and the Offset at the end
    device.write(fd.funcDict[function] + nd.noteDict[note] + ", ###, ###")
    # turn on the outputs
    device.write("OUTP1 ON")
    device.write("OUTP2 ON")

    # calculate the duration of the note
    # wait for this duration
    duration = ###

    # sleep for the calculated duration
    sleep(duration)

    # turn off the outputs
    device.write("OUTP1 OFF")
    device.write("OUTP2 OFF")

    # 0.05s break to get a little time between the notes
    sleep(0.05)

```

Ln: 4 Col: 12

Figure 14: playFunction example