Usage

First, import the module:

import sounddevice as sd

Playback

Assuming you have a NumPy array named myarray holding audio data with a sampling frequency of fs (in the most cases this will be 44100 or 48000 frames per second), you can play it back with play():

sd.play(myarray, fs)

This function returns immediately but continues playing the audio signal in the background. You can stop playback with stop():

sd.stop()

If you want to block the Python interpreter until playback is finished, you can use wait():

sd.wait()

If you know that you will use the same sampling frequency for a while, you can set it as default using default.samplerate:

sd.default.samplerate = fs

After that, you can drop the *samplerate* argument:

sd.play(myarray)

Note

If you don't specify the correct sampling frequency, the sound might be played back too slow or too fast!

Recording

To record audio data from your sound device into a NumPy array, you can use rec():

duration = 10.5 *# seconds*

myrecording = sd.rec(int(duration * fs), samplerate=fs, channels=2)

Again, for repeated use you can set defaults using default:

sd.default.samplerate = fs

sd.default.channels = 2

After that, you can drop the additional arguments:

myrecording = sd.rec(int(duration * fs))

This function also returns immediately but continues recording in the background. In the meantime, you can run other commands. If you want to check if the recording is finished, you should use wait():

sd.wait()

If the recording was already finished, this returns immediately; if not, it waits and returns as soon as the recording is finished.

By default, the recorded array has the data type 'float32' (see default.dtype), but this can be changed with the *dtype* argument:

myrecording = sd.rec(int(duration * fs), dtype='float64')

Simultaneous Playback and Recording

To play back an array and record at the same time, you can use playrec():

myrecording = sd.playrec(myarray, fs, channels=2)

The number of output channels is obtained from myarray, but the number of input channels still has to be specified.

Again, default values can be used:

sd.default.samplerate = fs

sd.default.channels = 2

myrecording = sd.playrec(myarray)

In this case the number of output channels is still taken from myarray (which may or may not have 2 channels), but the number of input channels is taken from default.channels.

Device Selection

In many cases, the default input/output device(s) will be the one(s) you want, but it is of course possible to choose a different device. Use query_devices() to get a list of supported devices. The same list can be obtained from a terminal by typing the command

python3 -m sounddevice

You can use the corresponding device ID to select a desired device by assigning to default.device or by passing it as *device* argument to play(), Stream() etc.

Instead of the numerical device ID, you can also use a space-separated list of case-insensitive substrings of the device name (and the host API name, if needed). See default.device for details.

import sounddevice as sd

sd.default.samplerate = 44100

sd.default.device = 'digital output'

sd.play(myarray)

Callback Streams

The aforementioned convenience functions play(), rec() and playrec() (as well as the related functions wait(), stop(), get_status() and get_stream()) are designed for small scripts and interactive use (e.g. in a Jupyter notebook). They are supposed to be simple and convenient, but their use cases are quite limited.

If you need more control (e.g. continuous recording, realtime processing, …), you should use the lower-level “stream” classes

(e.g. Stream, InputStream, RawInputStream), either with the “non-blocking” callback interface or with the “blocking” Stream.read() and Stream.write() methods, see Blocking Read/Write Streams.

As an example for the “non-blocking” interface, the following code creates a Stream with a callback function that obtains audio data from the input channels and simply forwards everything to the output channels (be careful with the output volume, because this might cause acoustic feedback if your microphone is close to your loudspeakers):

```
import sounddevice as sd

duration = 5.5  # seconds


def callback(indata, outdata, frames, time, status):
    if status:
        print(status)
    outdata[:] = indata


with sd.Stream(channels=2, callback=callback):
    sd.sleep(int(duration * 1000))
```

The same thing can be done with RawStream (NumPy doesn’t have to be installed):

```
import sounddevice as sd

duration = 5.5  # seconds


def callback(indata, outdata, frames, time, status):
```

```
    if status:
        print(status)
    outdata[:] = indata

with sd.RawStream(channels=2, dtype='int24', callback=callback):
    sd.sleep(int(duration * 1000))
```

Note

We are using 24-bit samples here for no particular reason (just because we can).

You can of course extend the callback functions to do arbitrarily more complicated stuff. You can also use streams without inputs (e.g. OutputStream) or streams without outputs (e.g. InputStream).

See Example Programs for more examples.

Blocking Read/Write Streams

Instead of using a callback function, you can also use the “blocking” methods Stream.read() and Stream.write() (and of course the corresponding methods in InputStream, OutputStream, RawStream, RawInputStream and RawOutputStream).