Playing and Recording Sound in Python

by Joska de Langen May 13, 2019 12 Comments basics

Mark as Completed

Share Share Email

Table of Contents

If you want to use Python to play or record sound, then you've come to the right place! In this tutorial, you'll learn how to play and record sound in Python using some of the most popular audio libraries. You will learn about the most straight-forward methods for playing and recording sound first, and then you'll learn about some libraries that offer some more functionality in exchange for a few extra lines of code.

By the end of this tutorial, you'll know how to:

- Play MP3 and WAV files, as well as a range of other audio formats
- Play NumPy and Python arrays containing sound
- Record sound using Python
- Save your recordings or audio files in a range of different file formats

For a comprehensive list of audio-related Python libraries, have a look at the wiki page on audio in Python.

Playing Audio Files

Below, you'll see how to play audio files with a selection of Python libraries. A few of these libraries let you play a range of audio formats, including MP3 and NumPy arrays. All of the libraries below let you play WAV files, some with a few more lines of code than others:

- **`playsound`** is the most straightforward package to use if you simply want to play a WAV or MP3 file. It offers no functionality other than simple playback.
- **`simpleaudio`** lets you play WAV files and NumPy arrays, and gives you options to check whether a file is still playing.
- **`winsound`** allows you to play WAV files or beep your speakers, but it works only on Windows.
- **`python-sounddevice`** and **`pyaudio`** provide bindings for the PortAudio library for cross-platform playback of WAV files.
- **`pydub`** requires **`pyaudio`** for audio playback, but with `ffmpeg` installed, it lets you play a large range of audio formats with only a few lines of code.

Let's have a look at these libraries for audio playback one by one.

`playsound`

`playsound` is a "pure Python, cross platform, single function module with no dependencies for playing sounds." With this module, you can play a sound file with a single line of code:

Python
```python
from playsound import playsound

playsound('myfile.wav')
```

The documentation of `playsound` states that it has been tested on WAV and MP3 files, but it may work for other file formats as well.

This library was last updated in June 2017. It seems to work well at the time of writing this article, but it's not clear whether it will still support newer Python releases.

simpleaudio

simpleaudio is a cross-platform library for playback of (mono and stereo) WAV files with no dependencies. The following code can be used to play a WAV file, and wait for the file to finish playing before terminating the script:

Python

```python
import simpleaudio as sa

filename = 'myfile.wav'
wave_obj = sa.WaveObject.from_wave_file(filename)
play_obj = wave_obj.play()
play_obj.wait_done()  # Wait until sound has finished playing
```

WAV files contain a **sequence of bits** representing the raw audio data, as well as **headers with metadata** in RIFF (Resource Interchange File Format) format.

For CD recordings, the industry standard is to store each audio sample (an individual audio datapoint relating to air pressure) as a **16-bit** value, at **44100** samples per second.

To reduce file size, it may be sufficient to store some recordings (for example of human speech) at a lower sampling rate, such as 8000 samples per second, although this does mean that higher sound frequencies may not be as accurately represented.

A few of the libraries discussed in this tutorial play and record `bytes` objects, whereas others use **NumPy arrays** to store raw audio data.

Both correspond to a sequence of data points that can be played back at a specified sample rate in order to play a sound. For a `bytes` object, each sample is stored as a set of two 8-bit values, whereas in a NumPy array, each element can contain a 16-bit value corresponding to a single sample.

An important difference between these two data types is that **`bytes` objects are immutable**, whereas **NumPy arrays are mutable**, making the latter more suitable for generating sounds and for more complex signal processing. For more information on how to work with NumPy, have a look at our NumPy tutorials.

`simpleaudio` allows you to play NumPy and Python arrays and `bytes` objects using `simpleaudio.play_buffer()`. Make sure you have NumPy installed for the following example to work, as well as `simpleaudio`. (With `pip` installed, you can do this by running `pip install numpy` from your console.)

For more information on how to use `pip` for installing packages, have a look at Pipenv: A Guide to the New Python Packaging Tool.

Below you'll see how to generate a NumPy array corresponding to a 440 Hz tone and play it back using `simpleaudio.play_buffer()`:

Python

```python
import numpy as np
import simpleaudio as sa

frequency = 440  # Our played note will be 440 Hz
fs = 44100  # 44100 samples per second
seconds = 3  # Note duration of 3 seconds

# Generate array with seconds*sample_rate steps, ranging between 0 and seconds
t = np.linspace(0, seconds, seconds * fs, False)

# Generate a 440 Hz sine wave
note = np.sin(frequency * t * 2 * np.pi)

# Ensure that highest value is in 16-bit range
audio = note * (2**15 - 1) / np.max(np.abs(note))
# Convert to 16-bit data
audio = audio.astype(np.int16)

# Start playback
play_obj = sa.play_buffer(audio, 1, 2, fs)

# Wait for playback to finish before exiting
play_obj.wait_done()
```

Next, let's see how you can use `winsound` to play WAV files on a Windows machine.

`winsound`

If you use Windows, you can use the built-in `winsound` module to access its basic sound-playing machinery. Playing a WAV file can be done in a few lines of code:

Python

```python
import winsound

filename = 'myfile.wav'
winsound.PlaySound(filename, winsound.SND_FILENAME)
```

`winsound` does not support playback of any files other than WAV files. It does allow you to beep your speakers using `winsound.Beep(frequency, duration)`. For example, you can beep a 1000 Hz tone for 100 milliseconds with the following code:

Python

```python
import winsound

winsound.Beep(1000, 100)  # Beep at 1000 Hz for 100 ms
```

Next, you'll learn how to use the `python-sounddevice` module for cross-platform audio playback.

`python-sounddevice`

As stated in its documentation, `python-sounddevice` "provides bindings for the PortAudio library and a few convenience functions to play and record NumPy arrays containing audio signals". In order to play WAV files, `numpy` and `soundfile` need to be installed, to open WAV files as NumPy arrays.

With `python-sounddevice`, `numpy`, and `soundfile` installed, you can now read a WAV file as a NumPy array and play it back:

Python

```
import sounddevice as sd
import soundfile as sf

filename = 'myfile.wav'
# Extract data and sampling rate from file
data, fs = sf.read(filename, dtype='float32')
sd.play(data, fs)
status = sd.wait()  # Wait until file is done playing
```

The line containing `sf.read()` extracts the raw audio data, as well as the sampling rate of the file as stored in its RIFF header, and `sounddevice.wait()` ensures that the script is only terminated after the sound finishes playing.

Next, we'll learn how to use `pydub` to play sound. With the right dependencies installed, it allows you to play a wide range of audio files, and it offers you more options for working with audio than `python-soundevice` does.

pydub

Although `pydub` can open and save WAV files without any dependencies, you need to have an audio playback package installed to play audio. `simpleaudio` is strongly recommended, but `pyaudio`, `ffplay`, and `avplay` are alternative options.

The following code can be used to play a WAV file with `pydub`:

Python

```
from pydub import AudioSegment
from pydub.playback import play

sound = AudioSegment.from_wav('myfile.wav')
play(sound)
```

In order to play back other audio types, such as MP3 files, `ffmpeg` or `libav` should be installed. Have a look at the documentation of `pydub` for instructions. As an alternative to the steps described in the documentation, `ffmpeg-python` provides bindings for `ffmpeg`, and can be installed using pip:

Shell

```
$ pip install ffmpeg-python
```

With `ffmpeg` installed, playing back an MP3 file requires only a small change in our earlier code:

Python

```
from pydub import AudioSegment
from pydub.playback import play

sound = AudioSegment.from_mp3('myfile.mp3')
play(sound)
```

Using the `AudioSegment.from_file(filename, filetype)` construction, you can play any type of audio file that `ffmpeg` supports. For example, you may play a WMA file using the ollowing:

Python

```
sound = AudioSegment.from_file('myfile.wma', 'wma')
```

In addition to playing back sound files, `pydub` lets you save audio in different file formats (more on this later), slice audio, calculate the length of audio files, fade in or out, and apply cross-fades.

`AudioSegment.reverse()` creates a copy of the AudioSegment that plays backwards, which the documentation describes as "useful for Pink Floyd, screwing around, and some audio processing algorithms."

pyaudio

pyaudio provides bindings for PortAudio, the cross-platform audio I/O library. This means that you can use `pyaudio` to play and record audio on a variety of platforms, including Windows, Linux, and Mac. With `pyaudio`, playing audio is done by writing to a `.Stream`:

Python

```python
import pyaudio
import wave

filename = 'myfile.wav'

# Set chunk size of 1024 samples per data frame
chunk = 1024

# Open the sound file
wf = wave.open(filename, 'rb')

# Create an interface to PortAudio
p = pyaudio.PyAudio()

# Open a .Stream object to write the WAV file to
# 'output = True' indicates that the sound will be played rather than recorded
stream = p.open(format = p.get_format_from_width(wf.getsampwidth()),
                channels = wf.getnchannels(),
                rate = wf.getframerate(),
                output = True)

# Read data in chunks
data = wf.readframes(chunk)

# Play the sound by writing the audio data to the stream
while data != '':
    stream.write(data)
    data = wf.readframes(chunk)

# Close and terminate the stream
stream.close()
p.terminate()
```

As you may have noticed, playing sounds with `pyaudio` is a bit more complex than playing sounds with the libraries you've seen earlier. This means that it may not be your first choice if you just want to play a sound effect in your Python application.

However, because `pyaudio` gives you more low-level control, it is possible to get and set parameters for your input and output devices, and to check your CPU load and input or output latency.

It also allows you to play and record audio in callback mode, where a specified callback function is called when new data is required for playback, or available for recording. These options make `pyaudio` a suitable library to use if your audio needs go beyond simple playback.

Now that you've seen how you can use a number of different libraries to play audio, it's time to see how you can use Python to record audio yourself.

Recording Audio

The python-sounddevice and pyaudio libraries provide ways to record audio with Python. `python-sounddevice` records to NumPy arrays and `pyaudio` records to `bytes` objects. Both of these can be stored as WAV files using the `scipy` and `wave` libraries, respectively.

python-sounddevice

python-sounddevice allows you to record audio from your microphone and store it as a NumPy array. This is a handy datatype for sound processing that can be converted to WAV format for storage using the `scipy.io.wavfile` module. Make sure to install the `scipy` module for the following example (`pip install scipy`). This automatically installs NumPy as one of its dependencies:

Python

```python
import sounddevice as sd
from scipy.io.wavfile import write

fs = 44100  # Sample rate
seconds = 3  # Duration of recording

myrecording = sd.rec(int(seconds * fs), samplerate=fs, channels=2)
sd.wait()  # Wait until recording is finished
write('output.wav', fs, myrecording)  # Save as WAV file
```

pyaudio

Earlier in this article, you learned how to play sounds by reading a `pyaudio.Stream()`. Recording audio can be done by writing to this stream instead:

Python

```python
import pyaudio
import wave

chunk = 1024  # Record in chunks of 1024 samples
sample_format = pyaudio.paInt16  # 16 bits per sample
channels = 2
fs = 44100  # Record at 44100 samples per second
seconds = 3
filename = "output.wav"

p = pyaudio.PyAudio()  # Create an interface to PortAudio

print('Recording')

stream = p.open(format=sample_format,
                channels=channels,
                rate=fs,
                frames_per_buffer=chunk,
                input=True)

frames = []  # Initialize array to store frames

# Store data in chunks for 3 seconds
for i in range(0, int(fs / chunk * seconds)):
    data = stream.read(chunk)
    frames.append(data)

# Stop and close the stream
stream.stop_stream()
stream.close()
# Terminate the PortAudio interface
p.terminate()

print('Finished recording')

# Save the recorded data as a WAV file
wf = wave.open(filename, 'wb')
wf.setnchannels(channels)
wf.setsampwidth(p.get_sample_size(sample_format))
wf.setframerate(fs)
wf.writeframes(b''.join(frames))
wf.close()
```

Now that you've seen how to record audio with `python-sounddevice` and `pyaudio`, you'll learn how to convert your recording (or any other audio files) to a range of different audio formats.

Saving and Converting Audio

You saw earlier that you can use the `scipy.io.wavfile` module to store NumPy arrays as WAV files. The `wavio` module similarly lets you convert between WAV files and NumPy arrays. If you want to store your audio in a different file format, `pydub` and `soundfile` come in handy, as they allow you to read and write a range of popular file formats (such as MP3, FLAC, WMA and FLV).

`wavio`

This module depends on `numpy` and lets you read WAV files as NumPy arrays, and save NumPy arrays as WAV files.

To save a NumPy array as a WAV file, you can use `wavio.write()`:

Python
```python
import wavio

wavio.write("myfile.wav", my_np_array, fs, sampwidth=2)
```

In this example, `my_np_array` is a NumPy array containing audio, `fs` is the sample rate of the recording (usually 44100 or 44800 Hz), and `sampwidth` is the sampling width of the audio (the number of bytes per sample, typically 1 or 2 bytes).

`soundfile`

The `soundfile` library can read and write all file formats supported by `libsndfile`. Although it can't play back audio, it allows you to convert audio from and to FLAC, AIFF, and a few audio formats that are less common . To convert a WAV file to FLAC, you can use the following code:

Python
```python
import soundfile as sf

# Extract audio data and sampling rate from file
data, fs = sf.read('myfile.wav')
# Save as FLAC file at correct sampling rate
sf.write('myfile.flac', data, fs)
```

Similar code will work for converting between other file formats supported by `libsndfile`.

`pydub`

`pydub` lets you save audio in any format that `ffmpeg` supports, which includes nearly all audio types you might encounter in your daily life. For example, you can convert your WAV file to MP3 with the following code:

Python
```python
from pydub import AudioSegment
sound = AudioSegment.from_wav('myfile.wav')

sound.export('myfile.mp3', format='mp3')
```

Using `AudioSegment.from_file()` is a more general way of loading audio files. For example, if you want to convert your file back from MP3 to WAV, you can do the following:

Python
```python
from pydub import AudioSegment
sound = AudioSegment.from_file('myfile.mp3', format='mp3')

sound.export('myfile.wav', format='wav')
```

This code should work for any audio file format that `ffmpeg` supports.

Comparison of Audio Libraries

Below is handy table that compares the functionality of the libraries discussed in this tutorial:

Library	Platform	Playback	Record	Convert	Dependencies
`playsound`	Cross-platform	WAV, MP3	-	-	None
`simpleaudio`	Cross-platform	WAV, array, `bytes`	-	-	None
`winsound`	Windows	WAV	-	-	None
`sounddevice`	Cross-platform	NumPy array	NumPy array	-	`numpy`, `soundfile`
`pydub`	Cross-platform	Any type supported by `ffmpeg`	-	Any type supported by `ffmpeg`	`simpleaudio`, `ffmpeg`
`pyaudio`	Cross-platform	`bytes`	`bytes`	-	`wave`
`wavio`	Cross-platform	-	-	WAV, NumPy array	`numpy`, `wave`
`soundfile`	Cross-platform	-	-	Any type supported by `libsndfile`	`CFFI`, `libsndfile`, `numpy`

The libraries that are included in this tutorial are chosen for their ease of use and popularity. For a more comprehensive list of audio libraries for Python, have a look at the wiki page on audio in Python.

Conclusion: Playing and Recording Sound in Python

In this tutorial, you learned how to use some of the most popular audio libraries to play and record audio in Python. You also saw how to save your audio in a range of different formats.

You are now able to:

- **Play** a large range of audio formats, including WAV, MP3 and NumPy arrays
- **Record** audio from your microphone to a NumPy or Python array
- **Store** your recorded audio a range of different formats, including WAV and MP3
- **Convert** your sound files to a range of different audio formats

You now have the information you need to help you decide which libraries to use to start working with audio in Python. Go forth, and develop some awesome audio applications!

Mark as Completed