PYTHON PROGRAMMING FOR AUDIO

Venkatakrishnan V K (vvaidyan@ur.rochester.edu)

ECE 477 – Computer Audition

University of Rochester
Department of Electrical and Computer Engineering

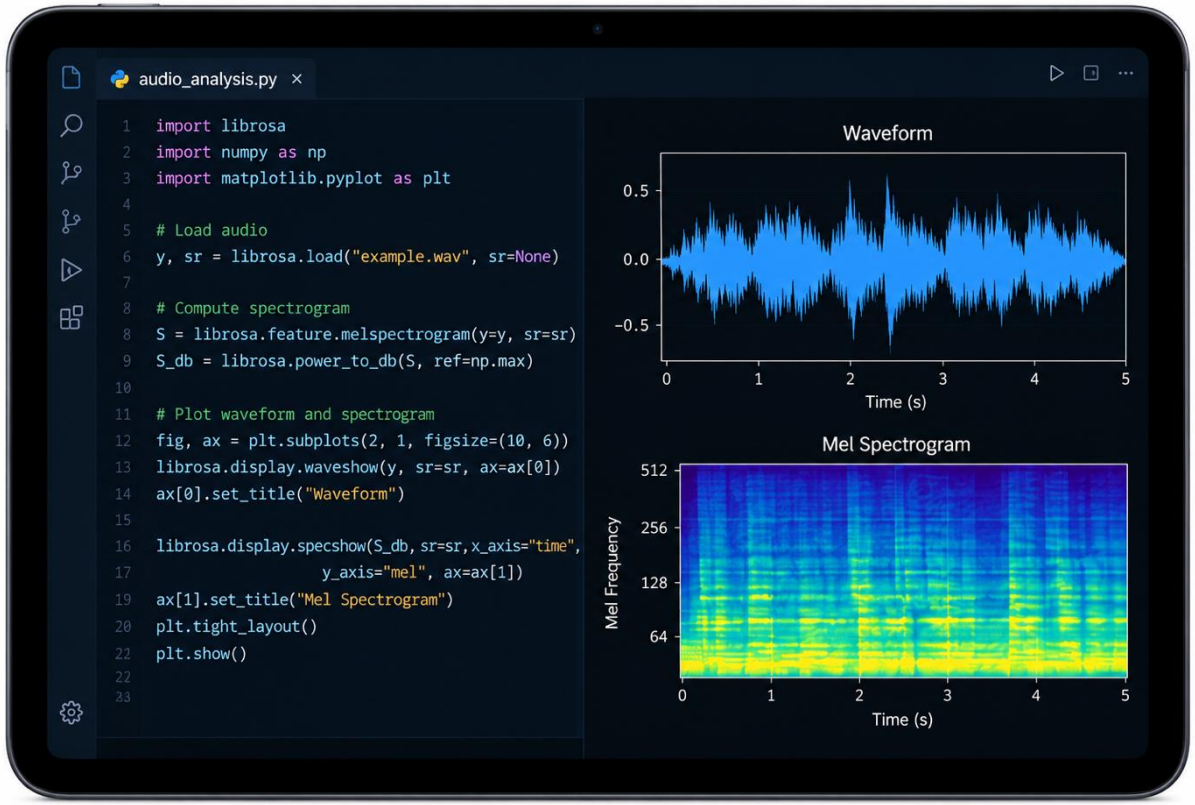
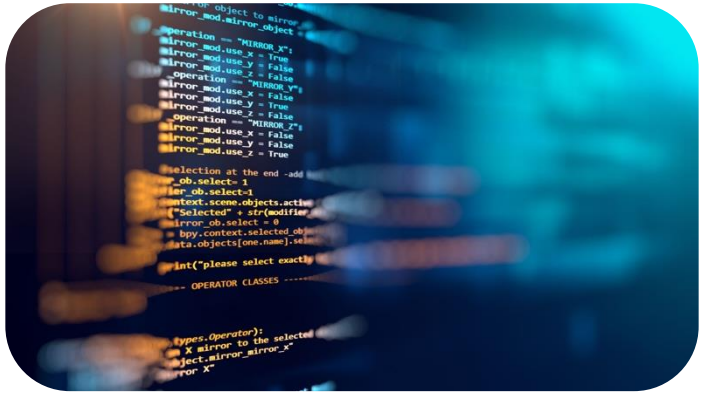


TABLE OF CONTENTS



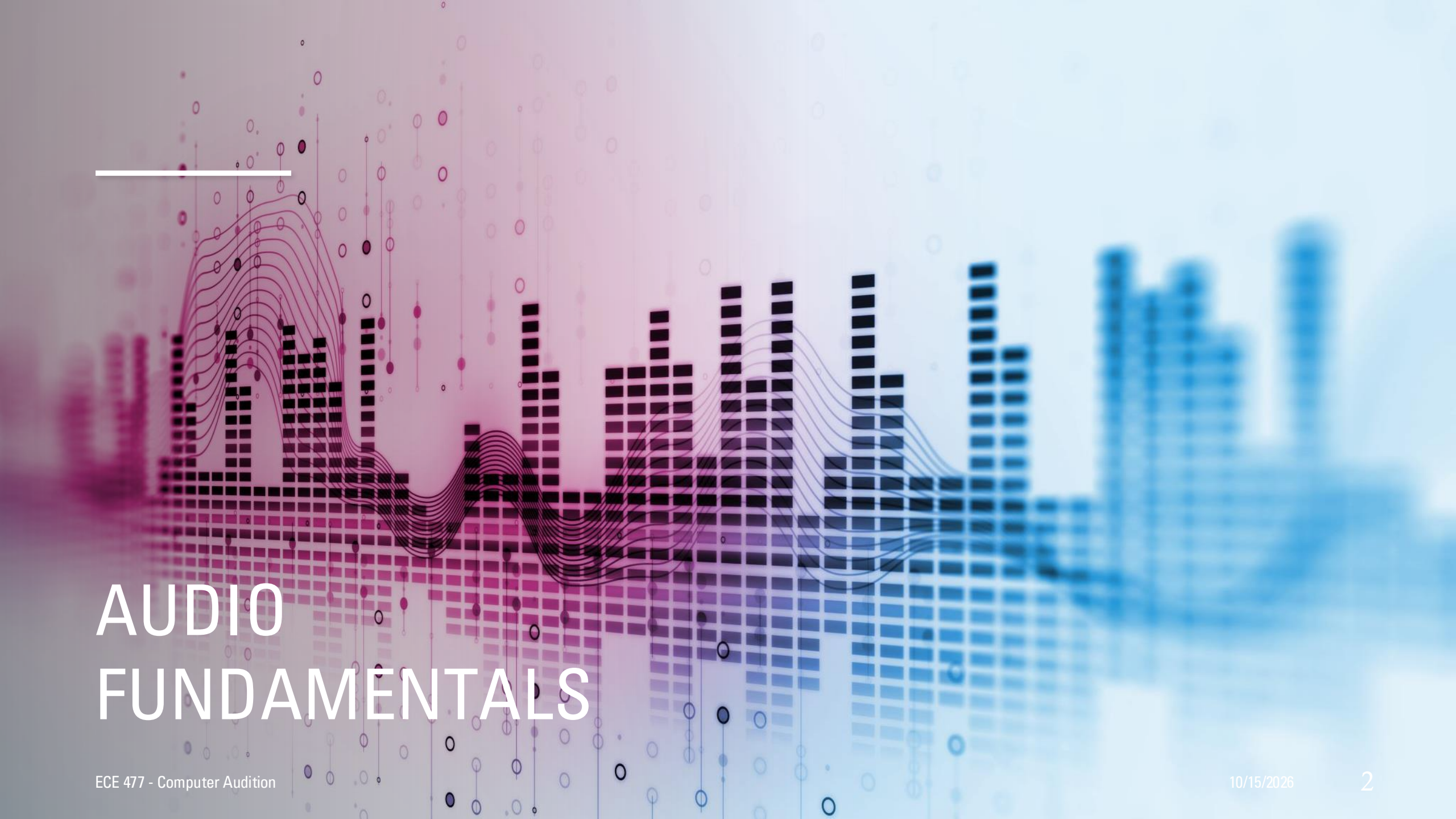
AUDIO FUNDAMENTALS



PYTHON LIBRARIES FOR AUDIO



AUDIO CODECS



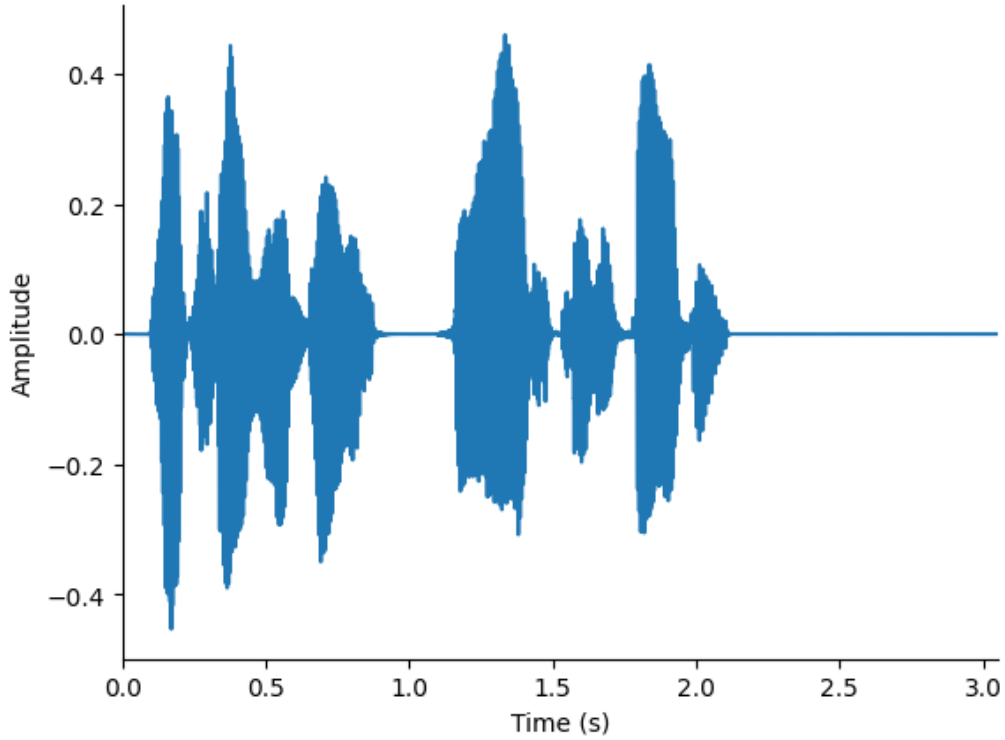
AUDIO FUNDAMENTALS

AUDIO WAVEFORMS

Introduction

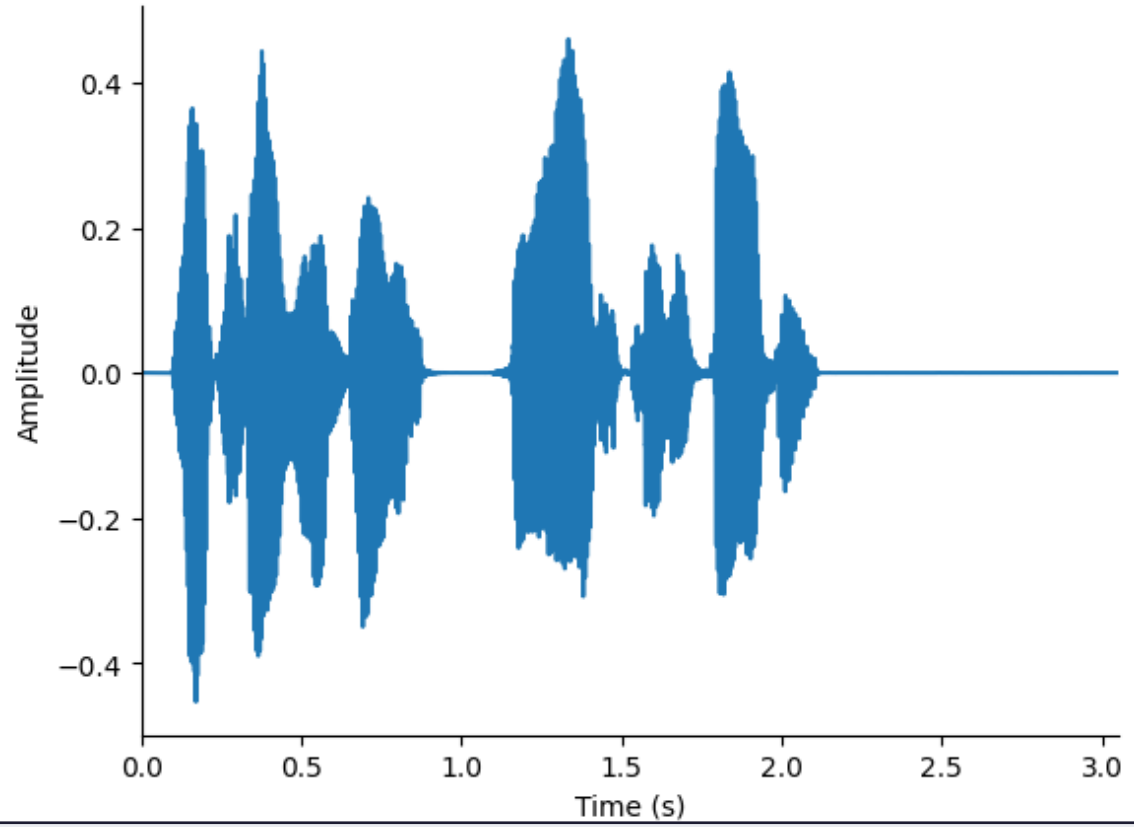
A visual representation of sound through alternating compressions and rarefactions





AUDIO WAVEFORMS

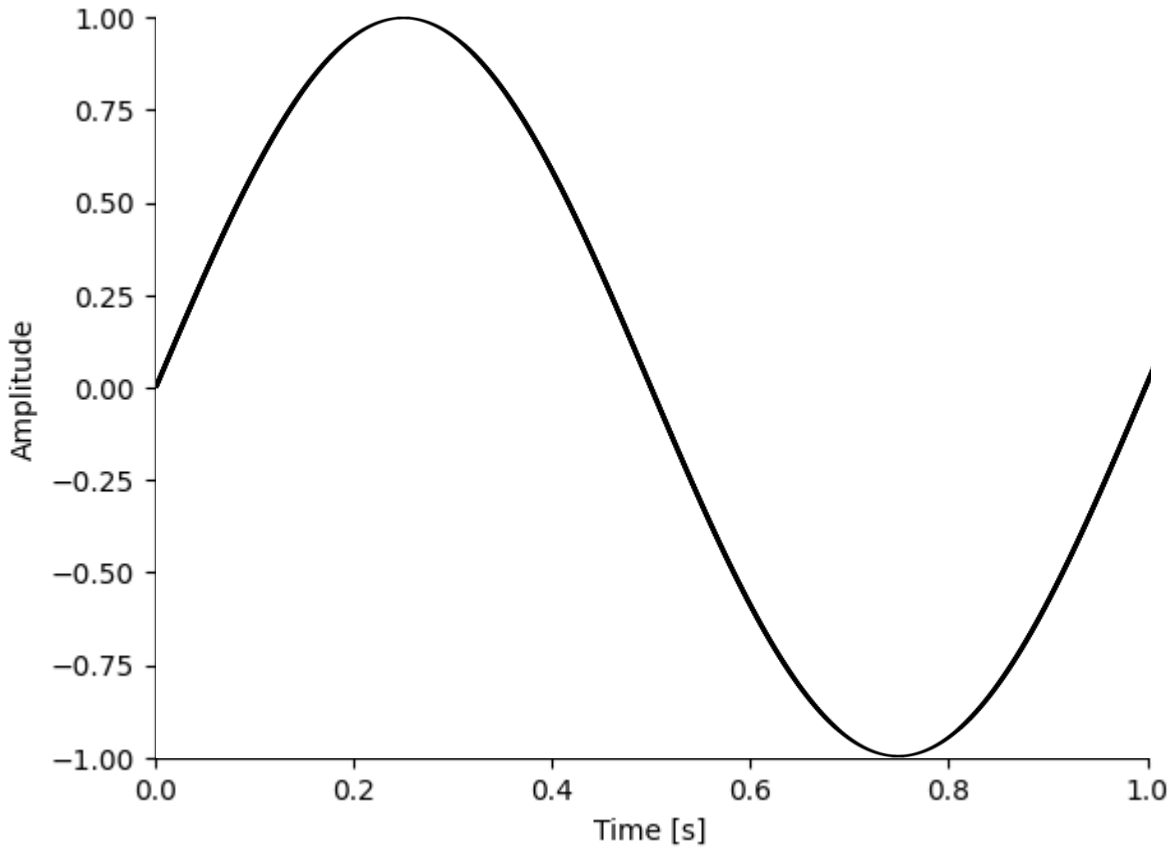
Introduction



A linear combination of sines and cosines!

SINE WAVE

Introduction



$$y = A\sin(2\pi f t + \phi_0)$$

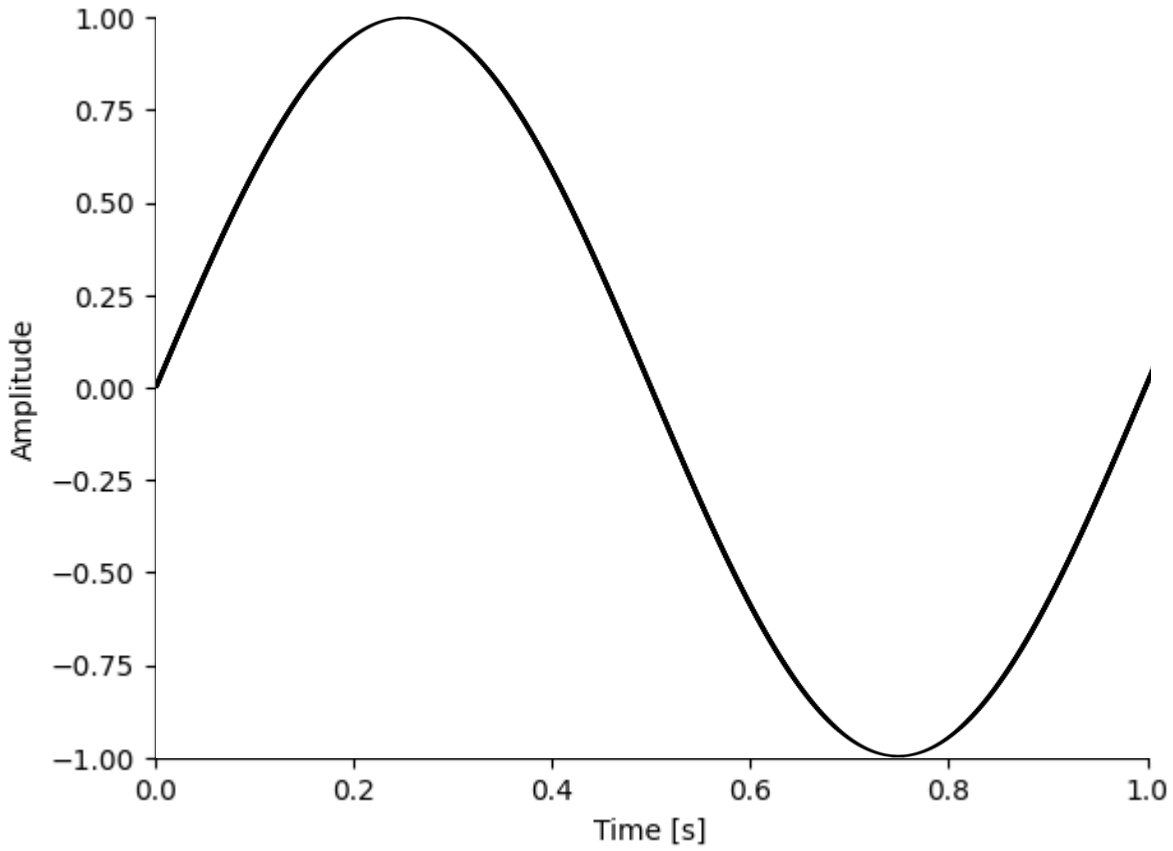
Amplitude
= 1.0

Frequency
= 1 Hz

Initial Phase
= 0 rad

SINE WAVE

Amplitude



$$y = A\sin(2\pi f t + \phi_0)$$

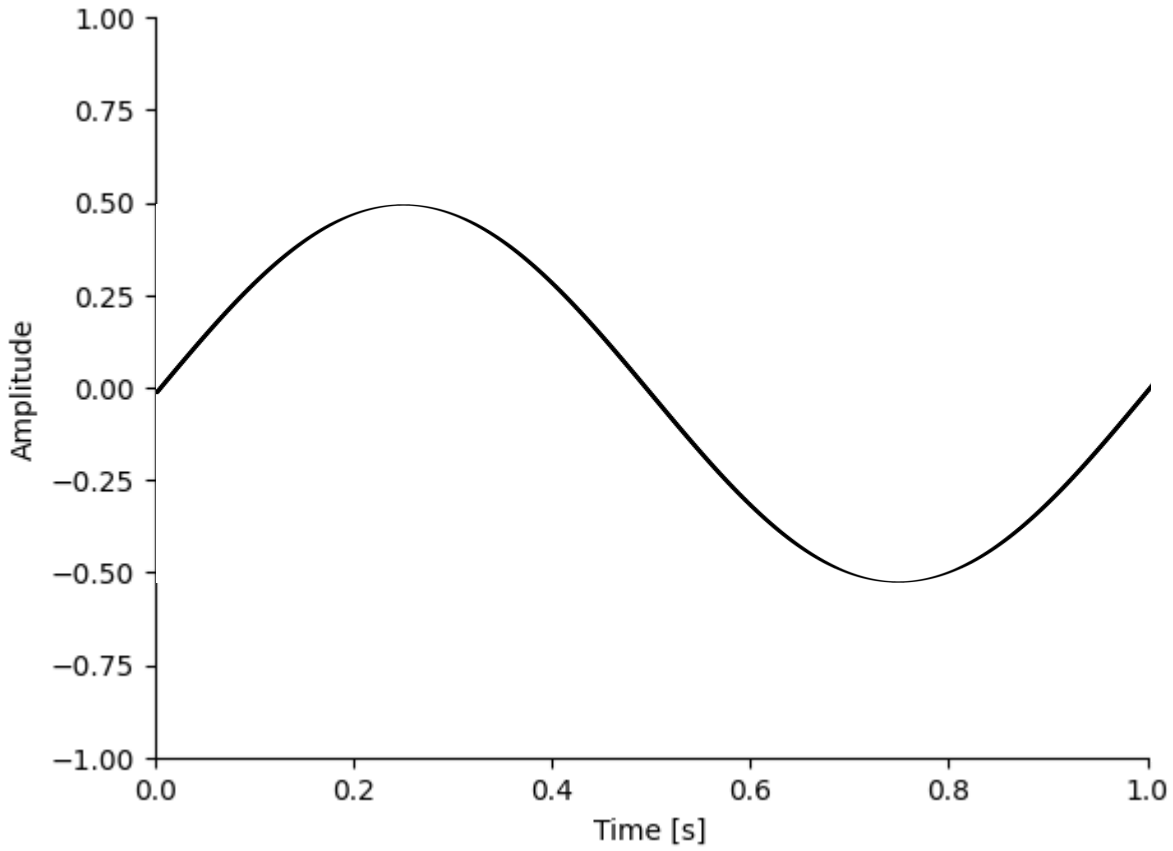
Amplitude
= 1.0

Frequency
= 1 Hz

Initial Phase
= 0 rad

SINE WAVE

Amplitude



$$y = A\sin(2\pi f t + \phi_0)$$

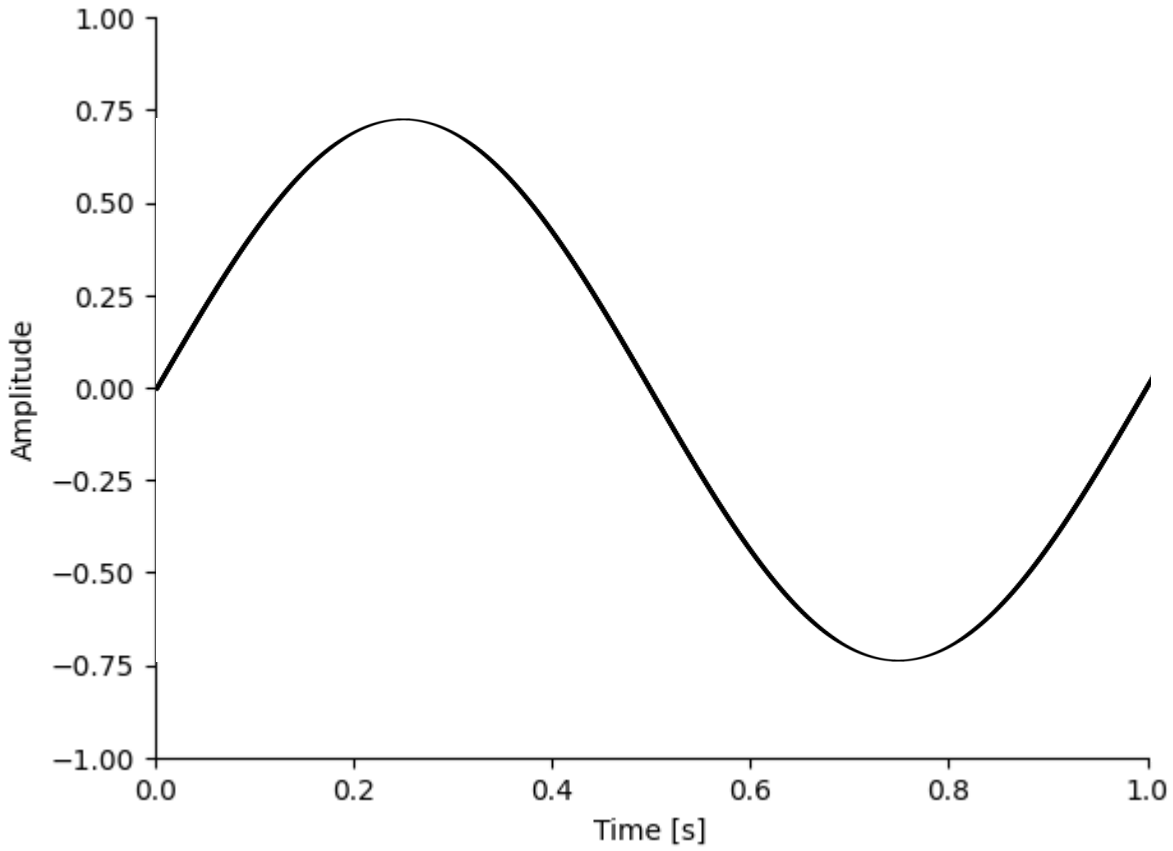
Amplitude
= 0.5

Frequency
= 1 Hz

Initial Phase
= 0 rad

SINE WAVE

Amplitude



$$y = A\sin(2\pi f t + \phi_0)$$

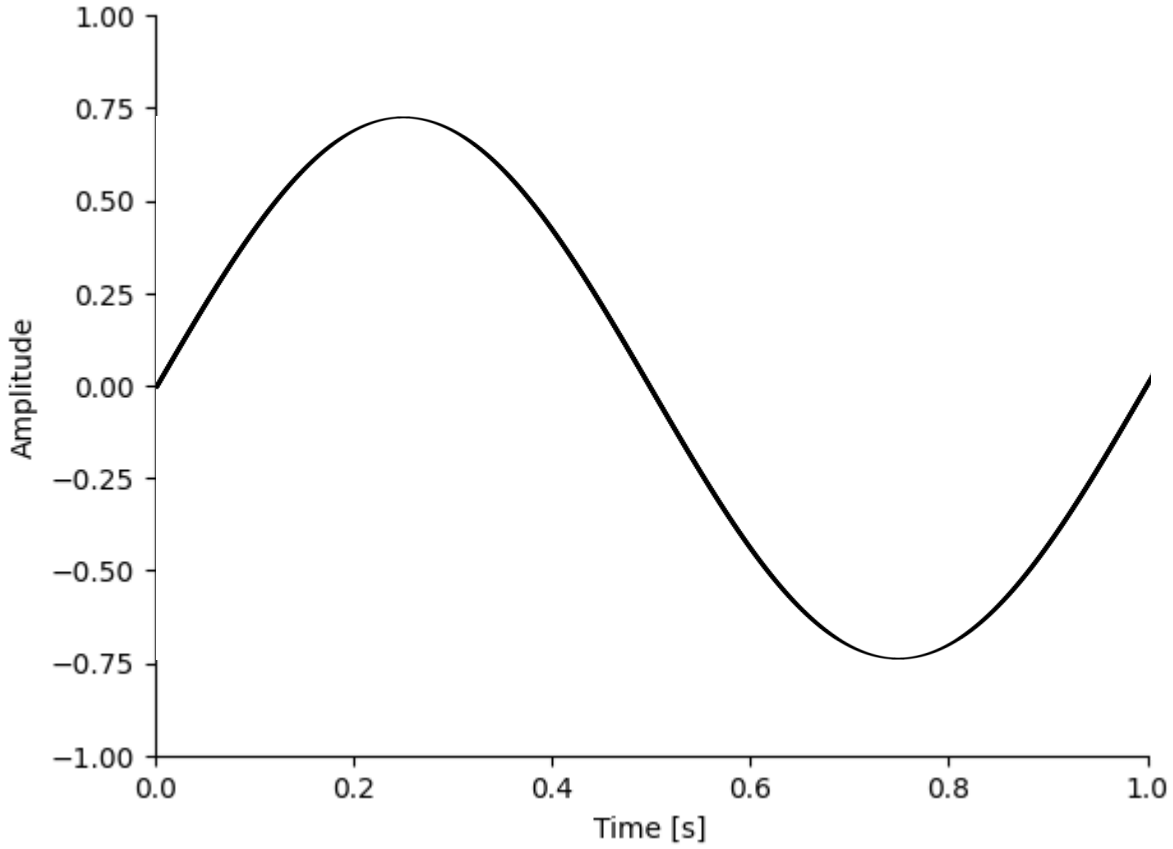
Amplitude
= 0.75

Frequency
= 1 Hz

Initial Phase
= 0 rad

SINE WAVE

Frequency



$$y = A\sin(2\pi f t + \phi_0)$$

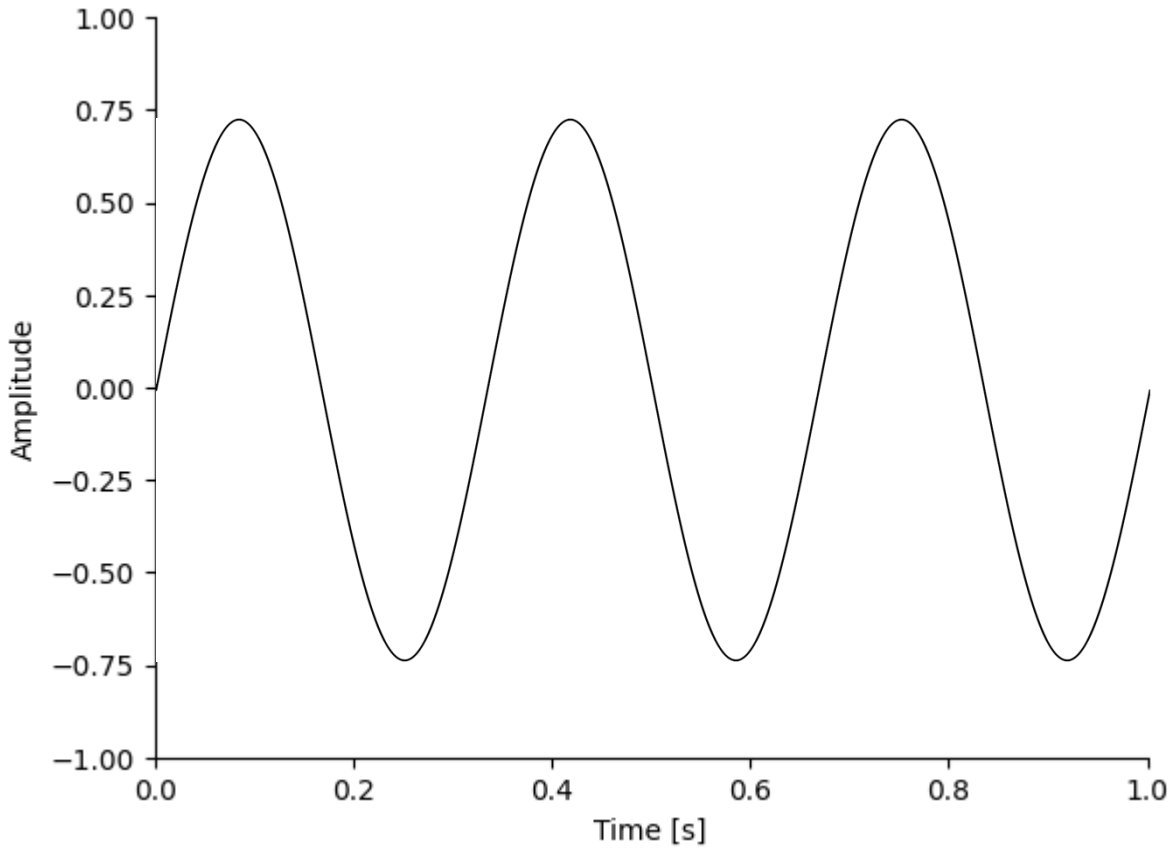
Amplitude
= 0.75

Frequency
= 1 Hz

Initial Phase
= 0 rad

SINE WAVE

Frequency



$$y = A\sin(2\pi f t + \phi_0)$$

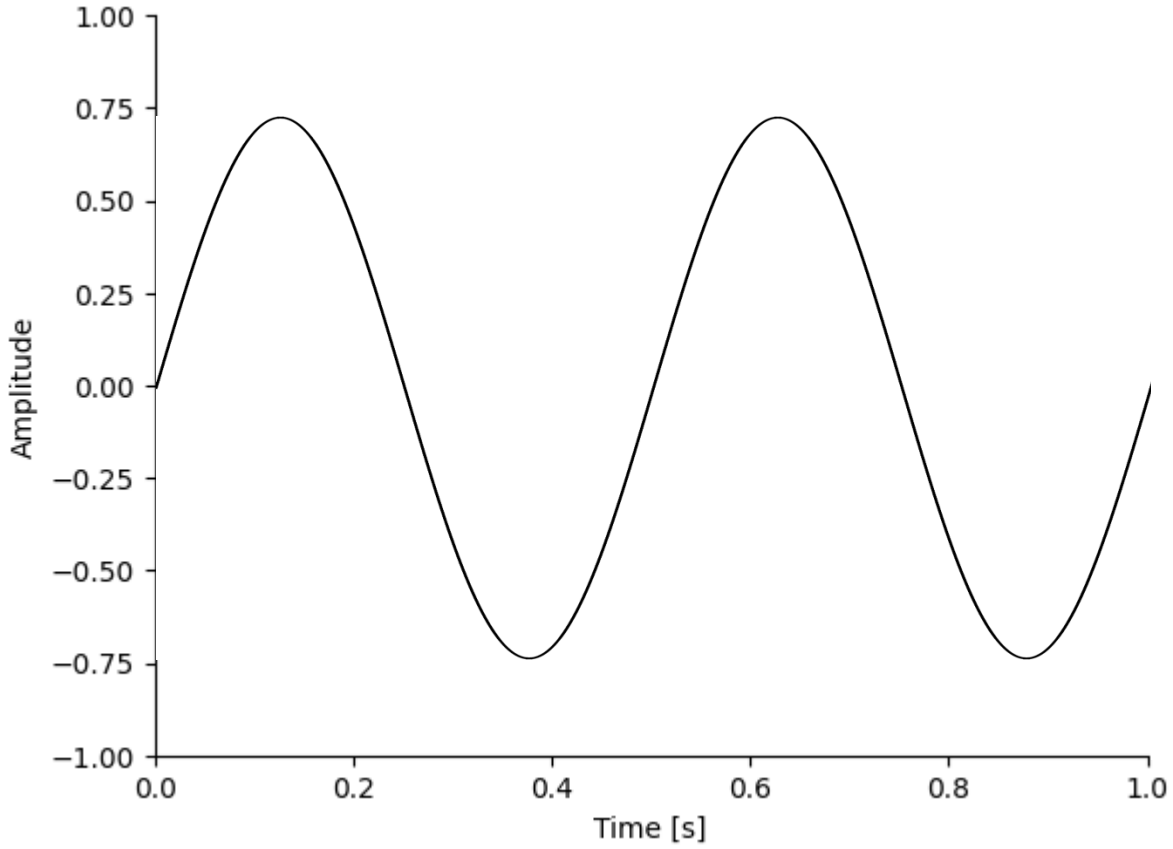
Amplitude
= 0.75

Frequency
= 3 Hz

Initial Phase
= 0 rad

SINE WAVE

Frequency



$$y = A\sin(2\pi f t + \phi_0)$$

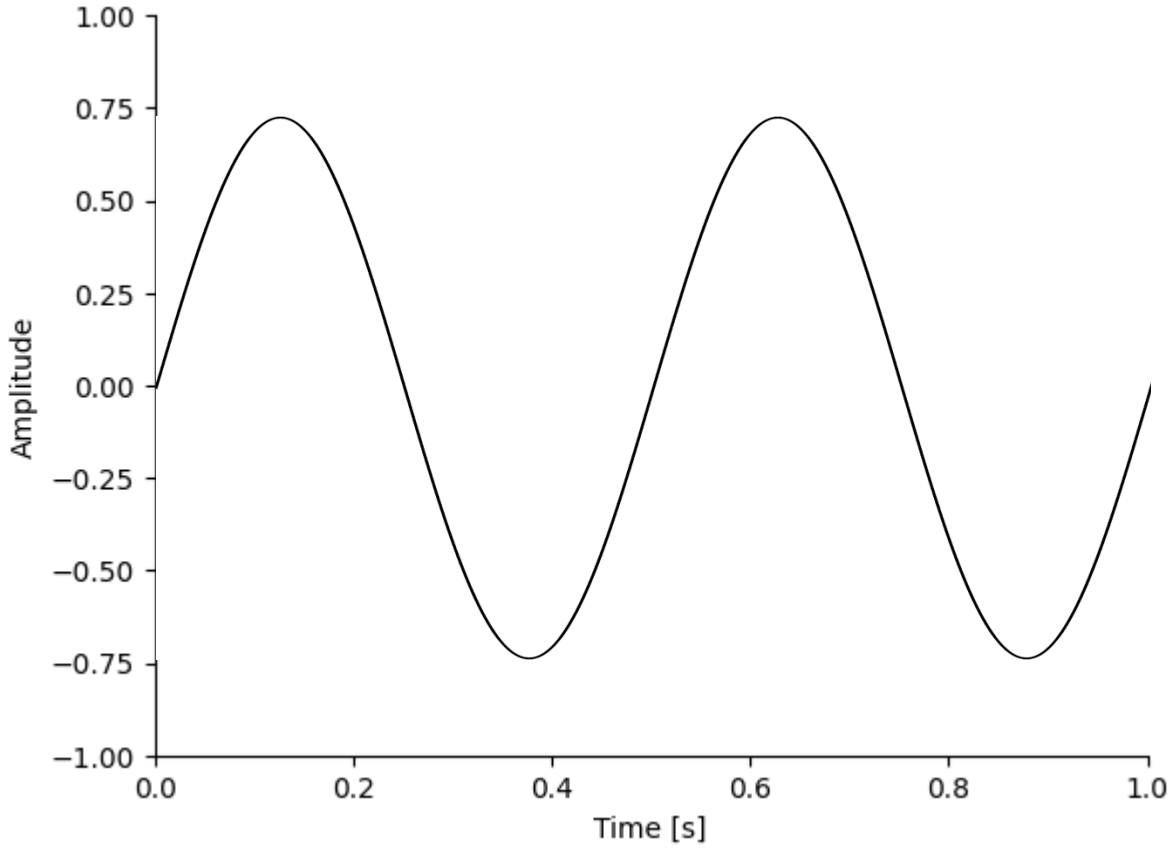
Amplitude
= 0.75

Frequency
= 2 Hz

Initial Phase
= 0 rad

SINE WAVE

Phase



$$y = A\sin(2\pi f t + \phi_0)$$

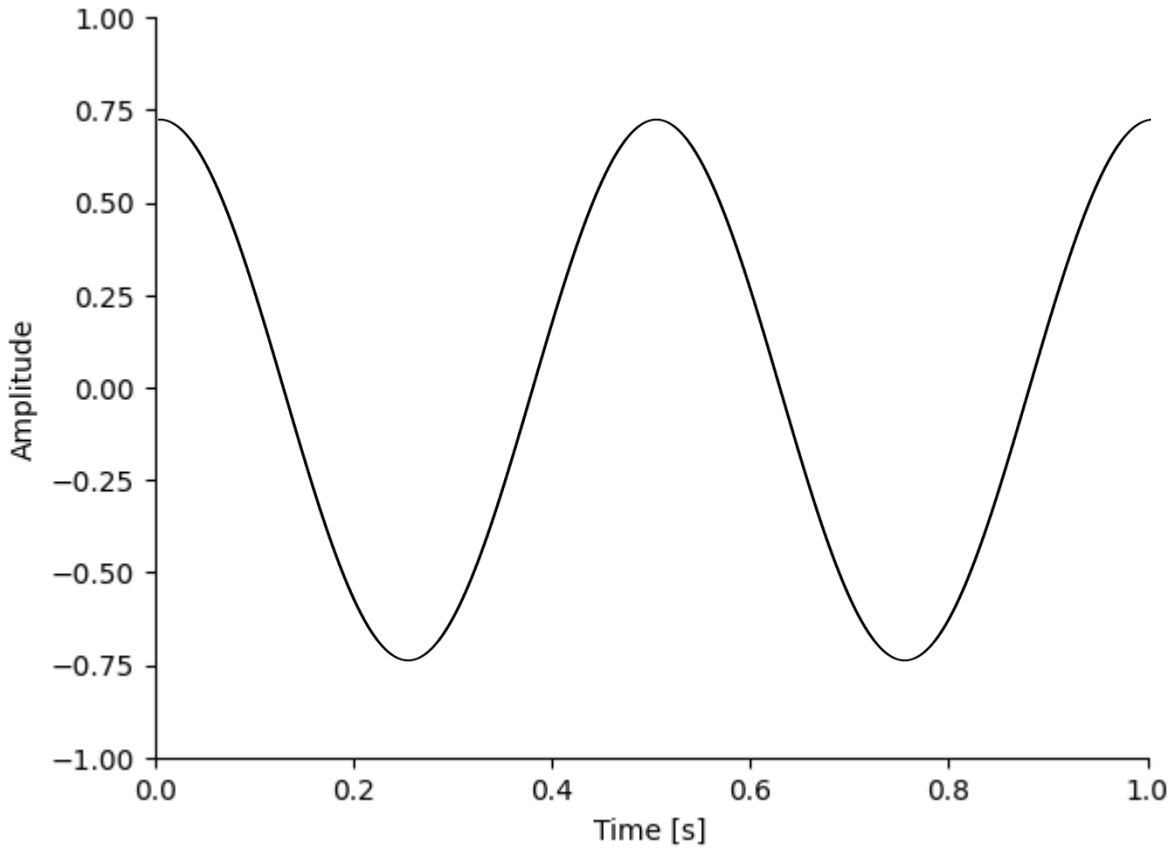
Amplitude
= 0.75

Frequency
= 2 Hz

Initial Phase
= 0 rad

SINE WAVE

Phase



$$y = A\sin(2\pi f t + \phi_0)$$

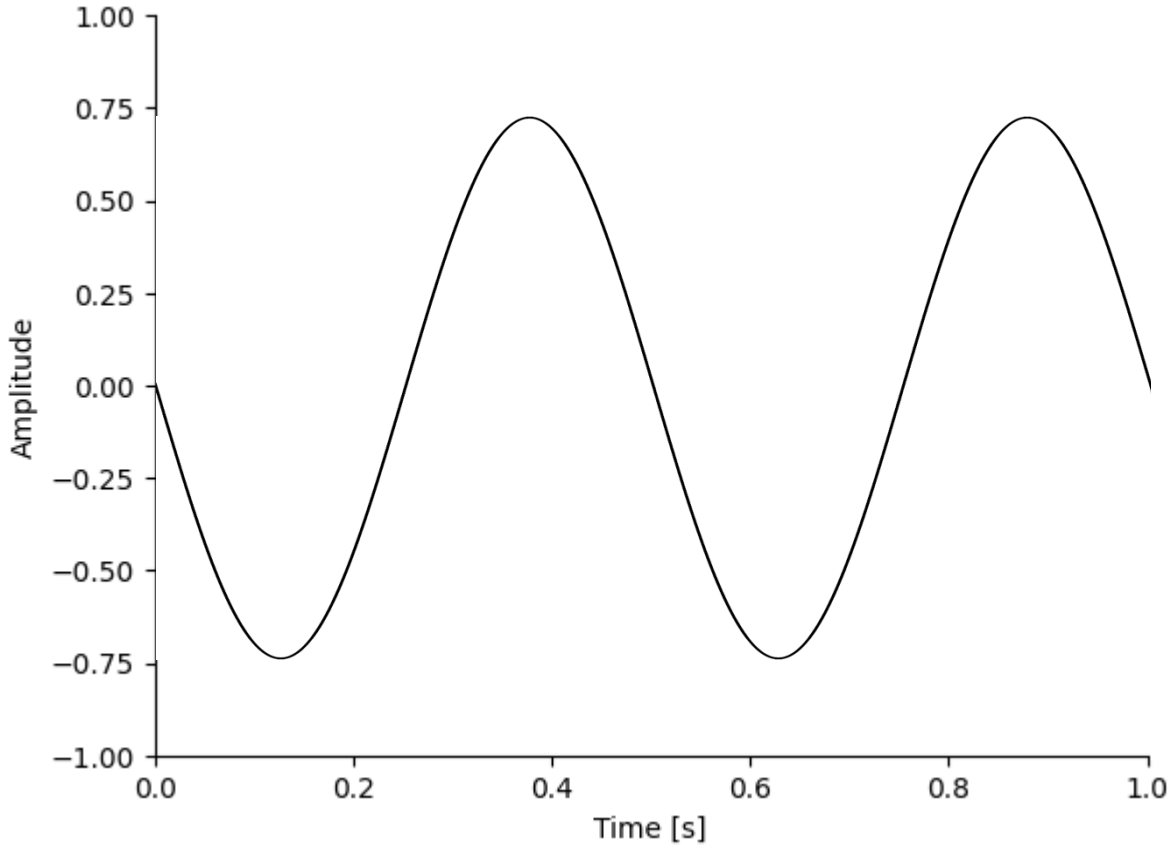
Amplitude
$= 0.75$

Frequency
$= 2$ Hz

Initial Phase
$= \pi/2$ **rad**

SINE WAVE

Phase



$$y = A\sin(2\pi f t + \phi_0)$$

Amplitude
= 0.75

Frequency
= 2 Hz

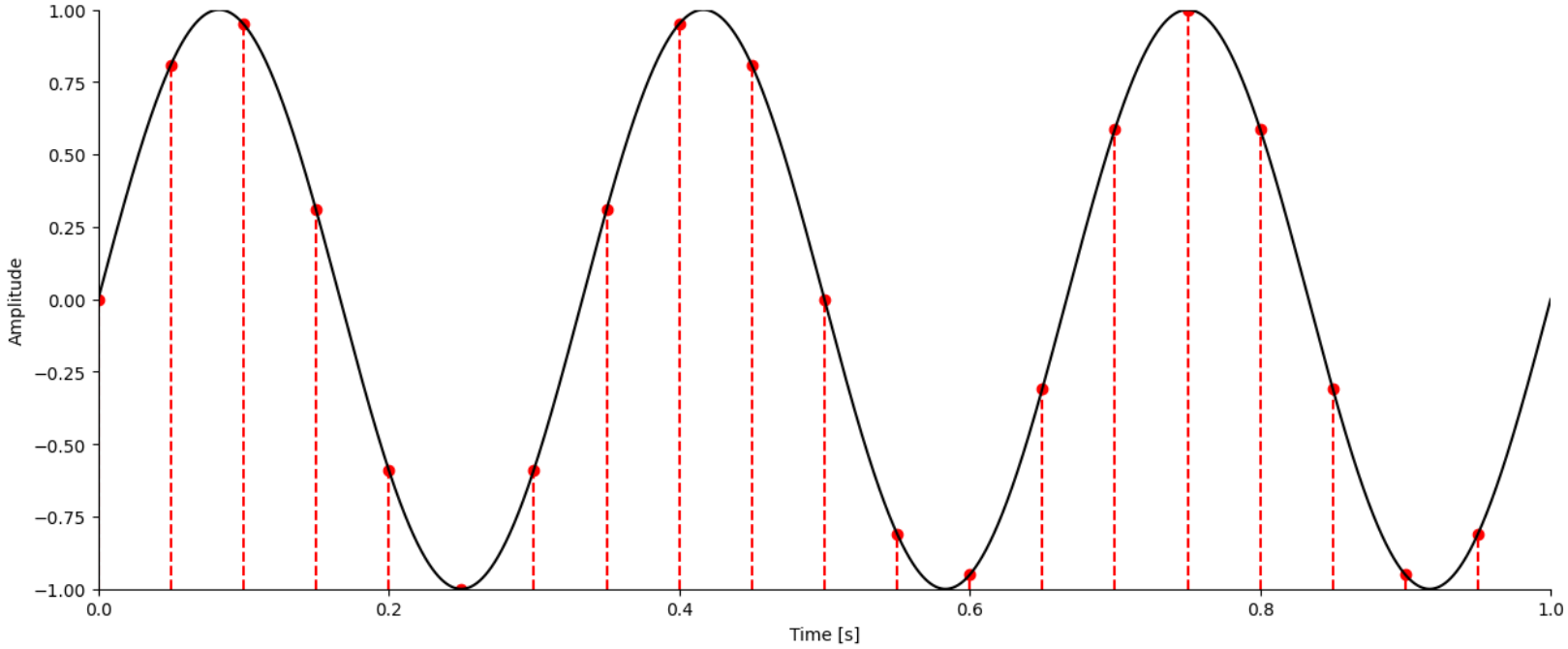
Initial Phase
= π rad

DIGITAL SIGNALS

Sampling

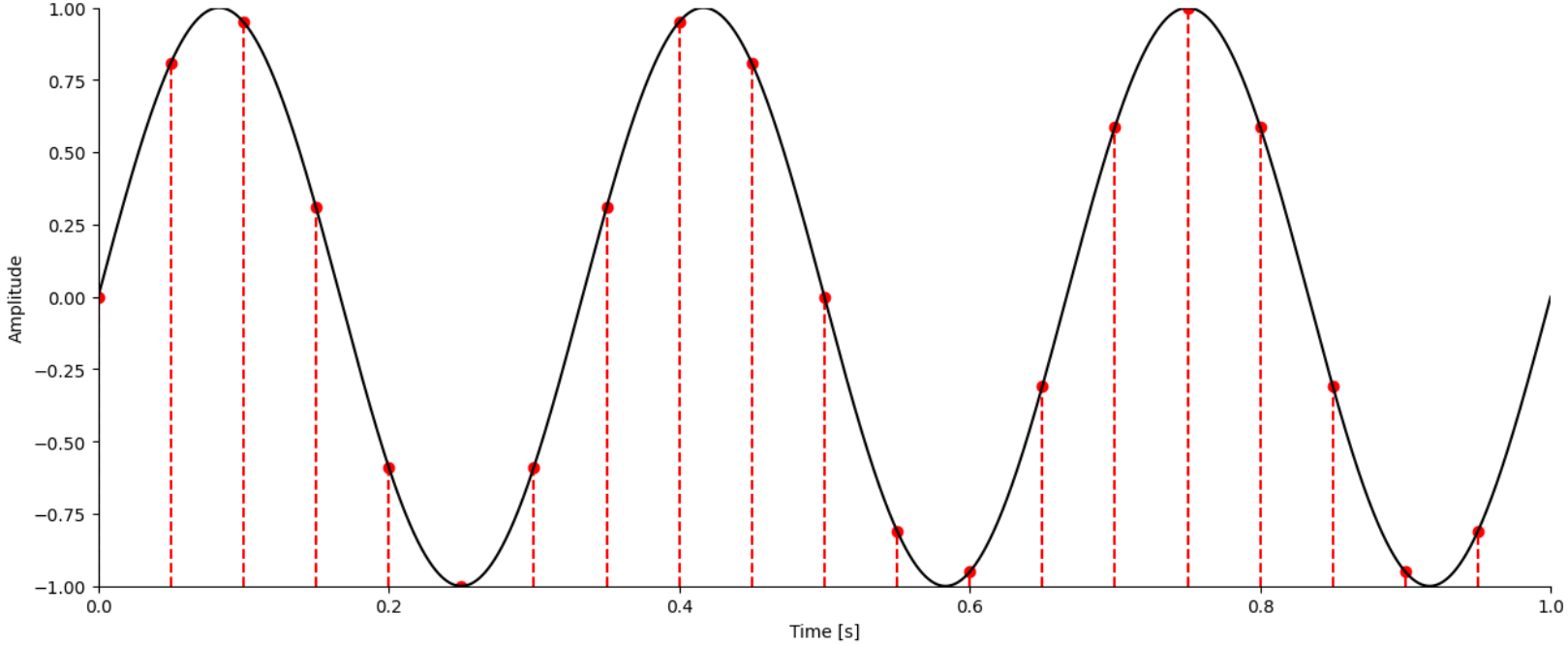
Real signals are continuous

Digital equipment requires discrete!



DIGITAL SIGNALS

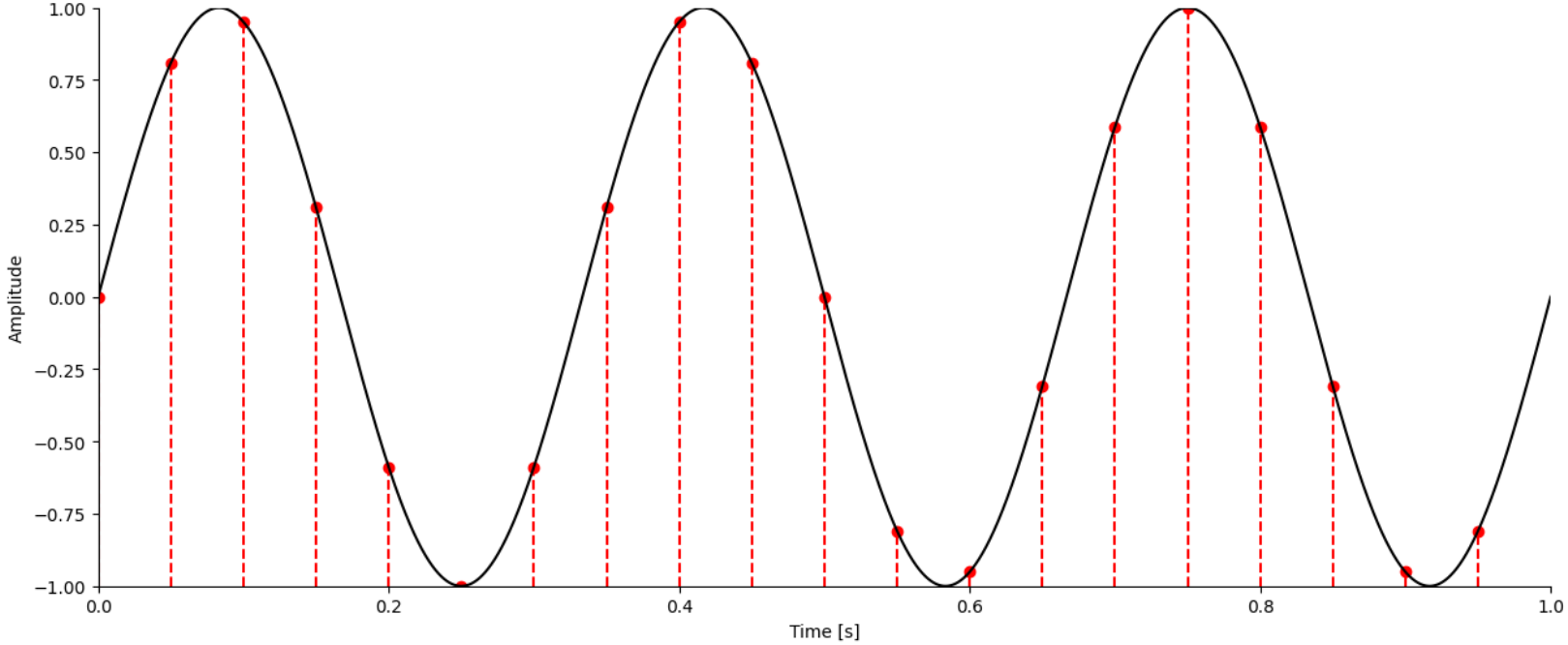
Sampling



The number of times a signal is sampled per second: **Sampling Rate**

DIGITAL SIGNALS

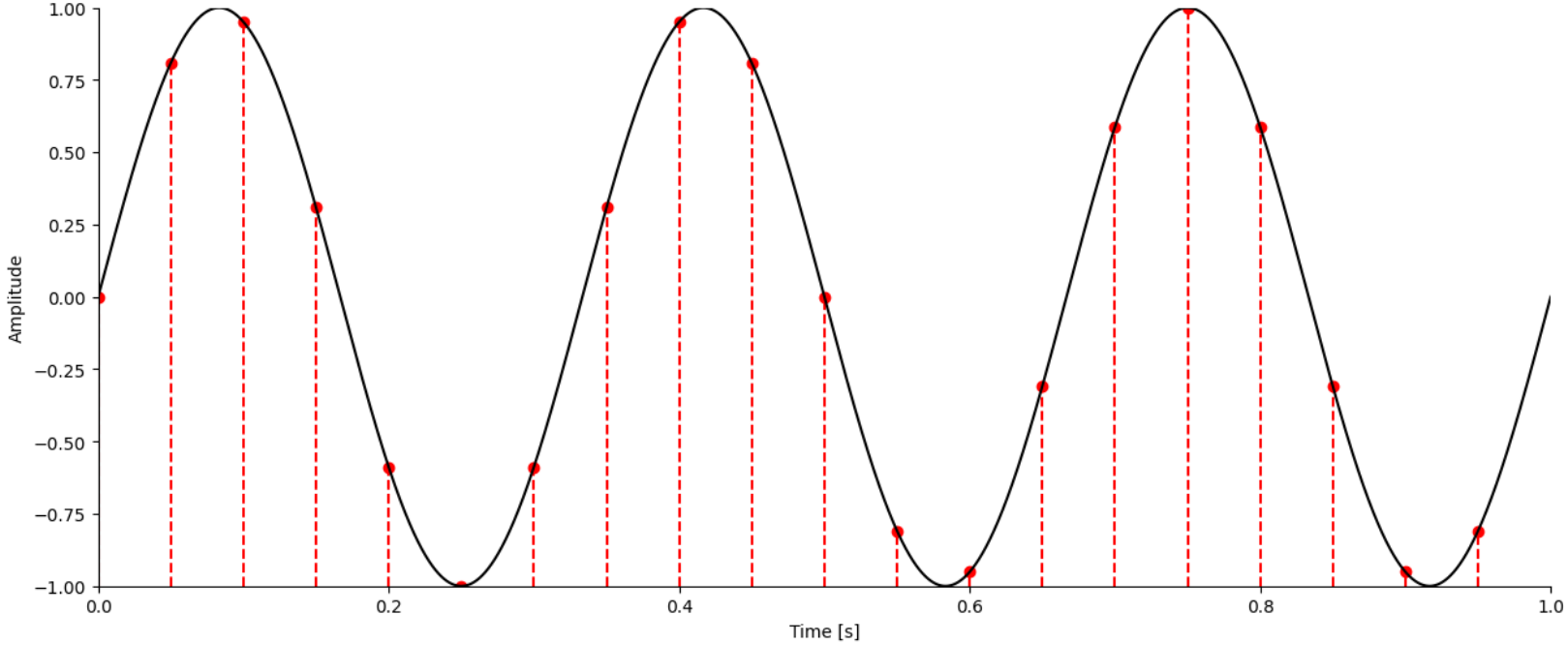
Sampling



Sampling Rate = ?

DIGITAL SIGNALS

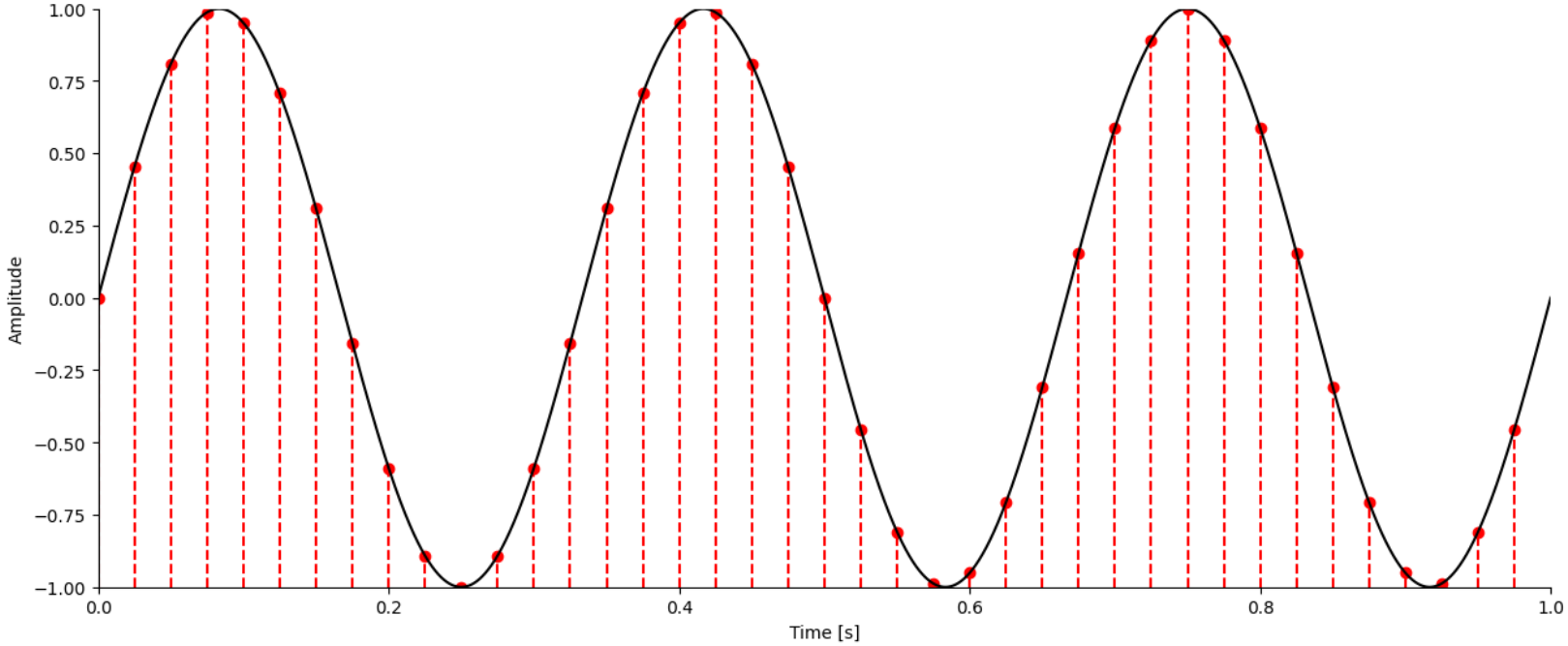
Sampling



Sampling Rate = 20 Hz

DIGITAL SIGNALS

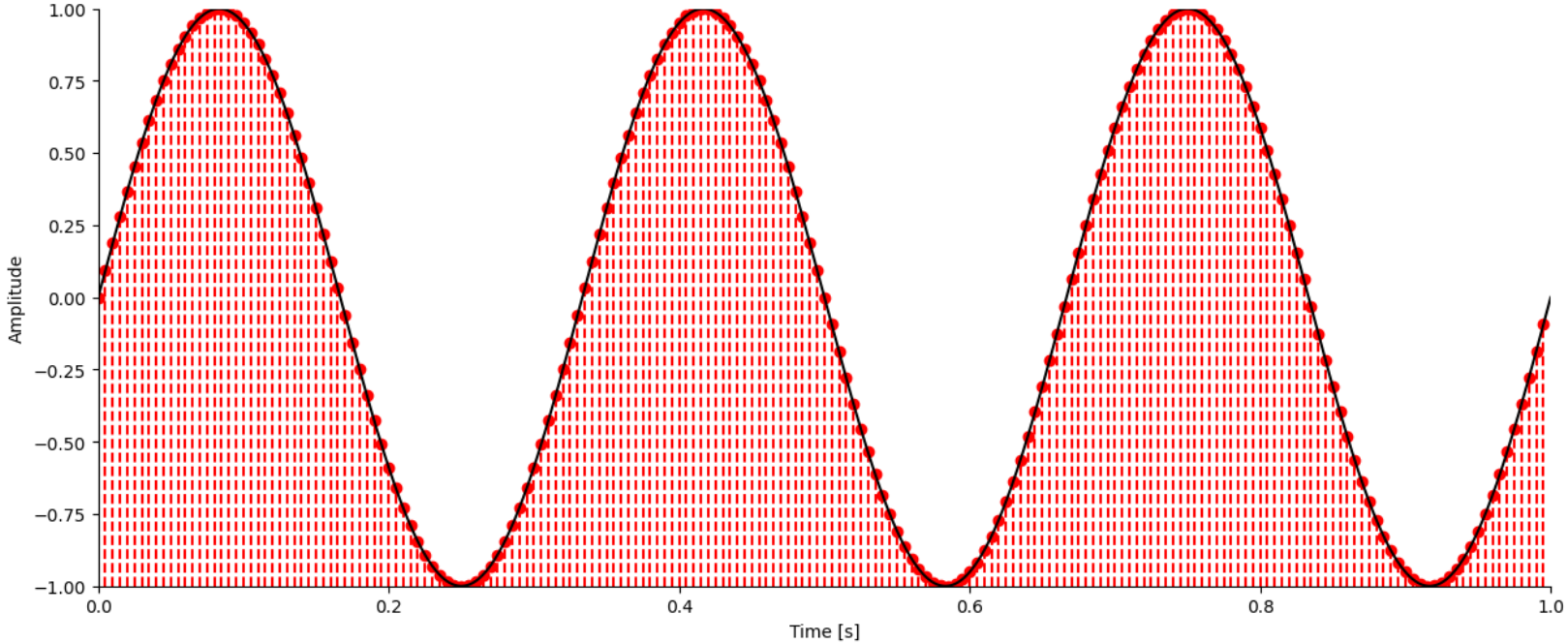
Sampling



Sampling Rate = 40 Hz

DIGITAL SIGNALS

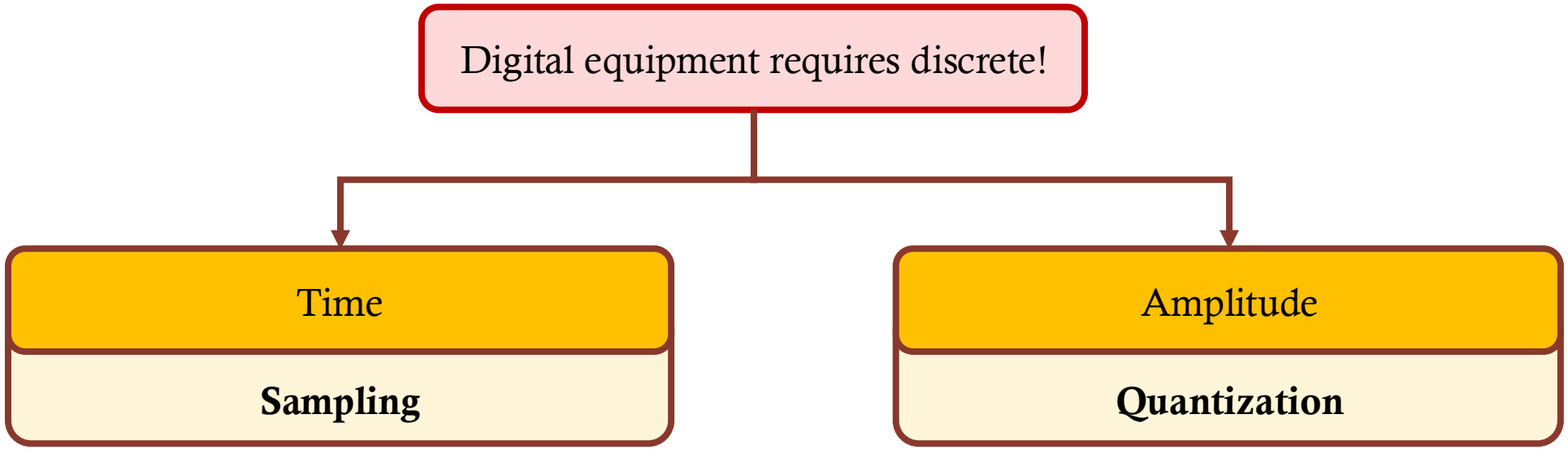
Sampling



Sampling Rate = 200 Hz

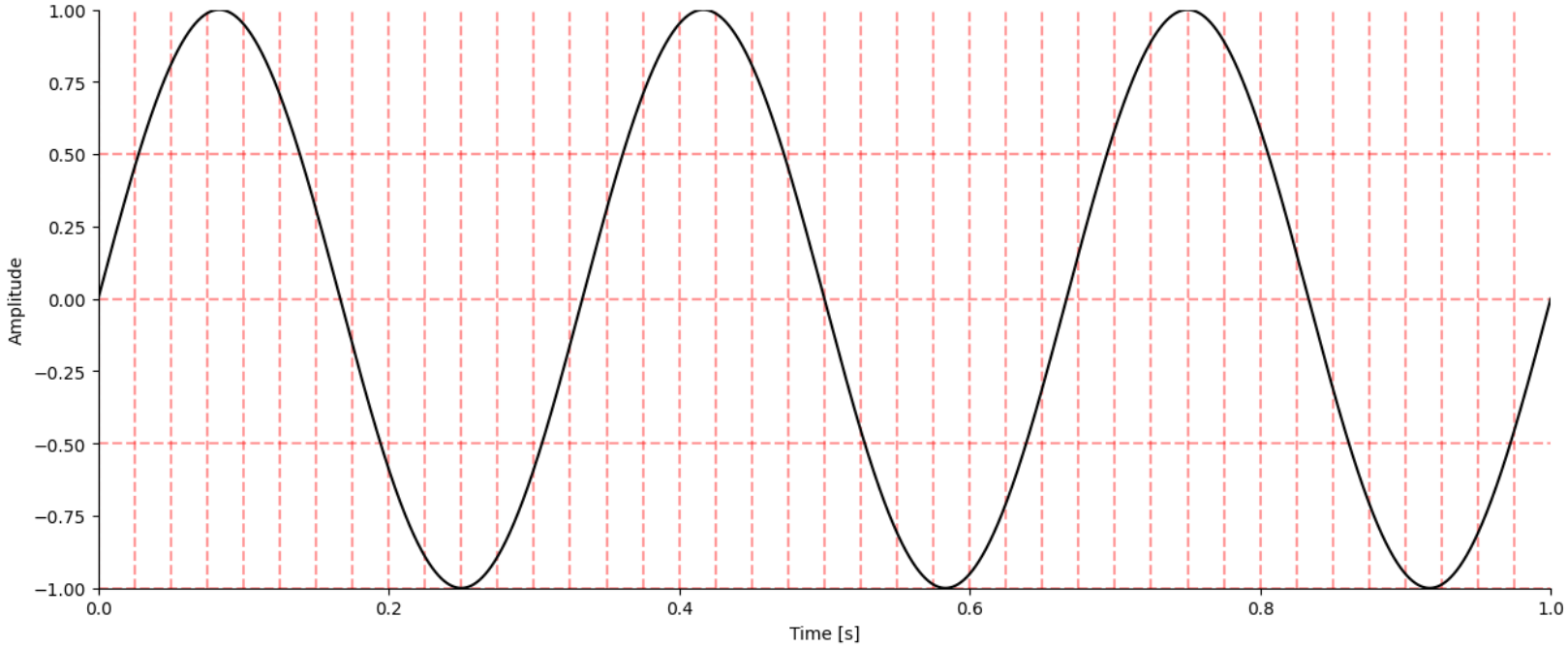
DIGITAL SIGNALS

Quantization: Bit Depth



DIGITAL SIGNALS

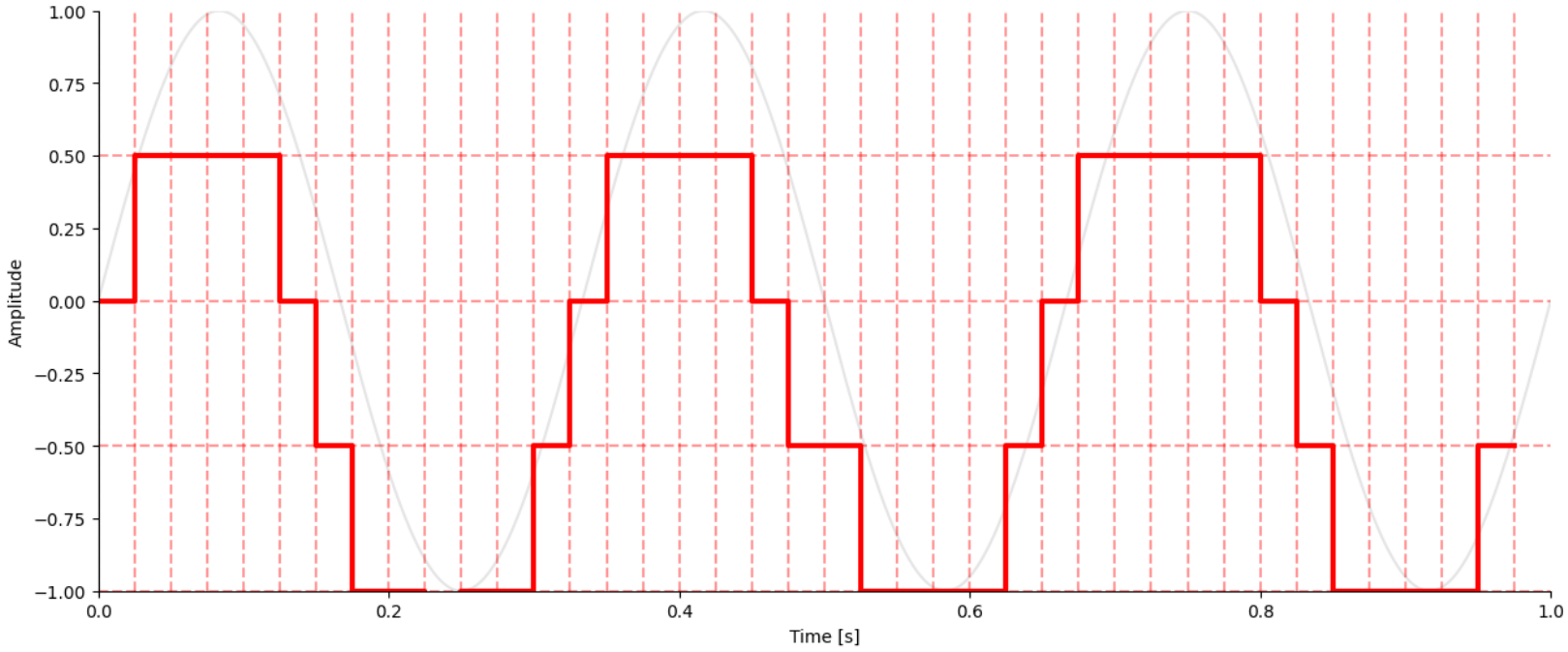
Quantization: Bit Depth



Bit Depth: **2 bits**

DIGITAL SIGNALS

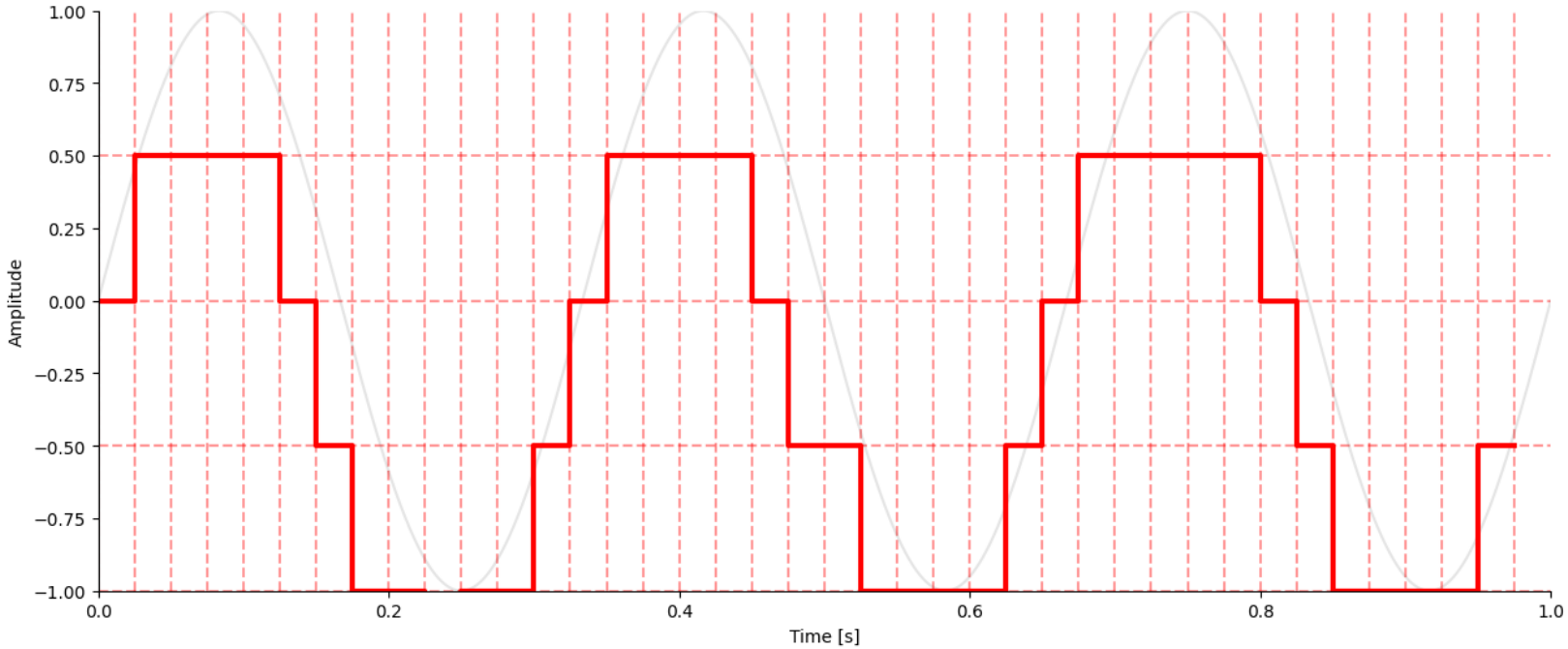
Quantization: Bit Depth



Bit Depth: **2 bits**

DIGITAL SIGNALS

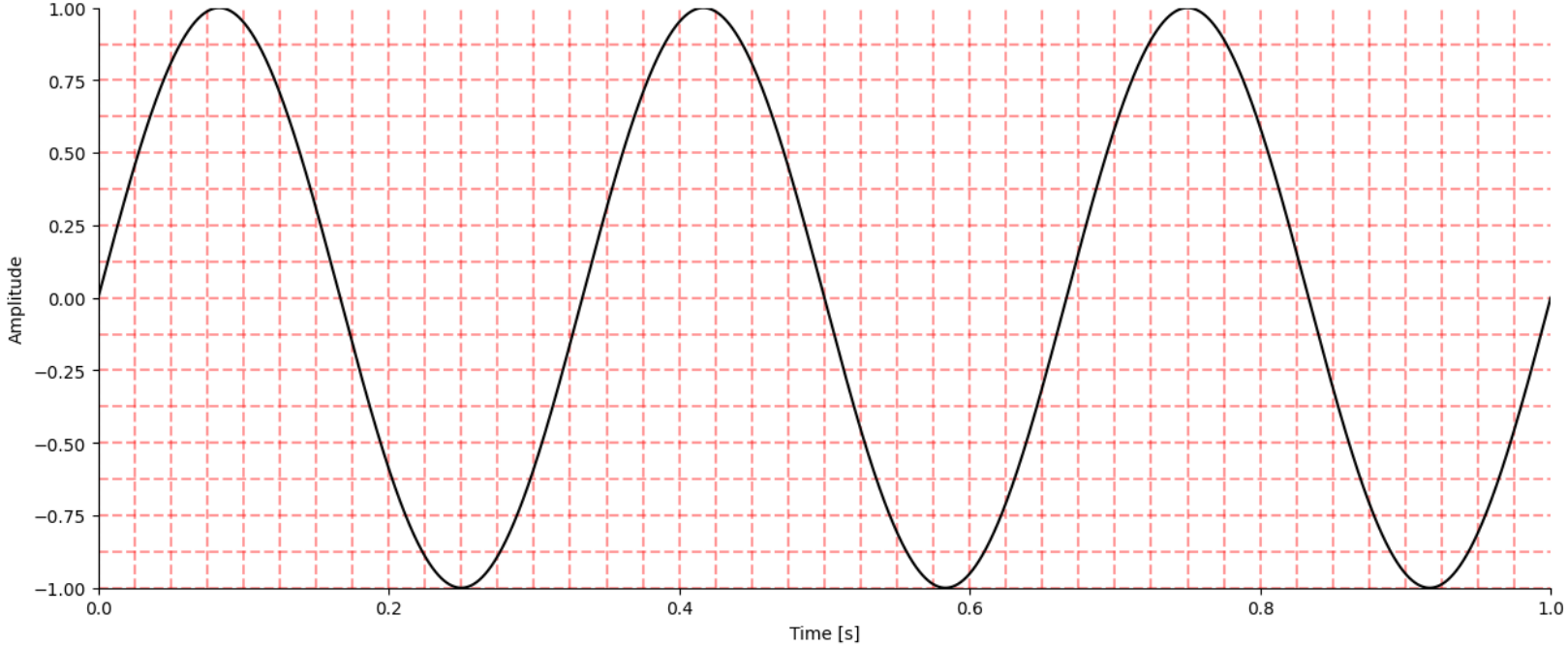
Quantization: Bit Depth




```
array([ 0. , 0.5, 0.5, 0.5, 0.5, 0. , -0.5, -1. , -1. , -1.5, -1. , -1. , -
0.5, 0. , 0.5, 0.5, 0.5, 0.5, 0. , -0.5, -0.5, -1. , -1. , -1. , -1. , -0.5,
0. , 0.5, 0.5, 0.5, 0.5, 0.5, 0. , -0.5, -1. , -1. , -1. , -1. , -0.5])
```

DIGITAL SIGNALS

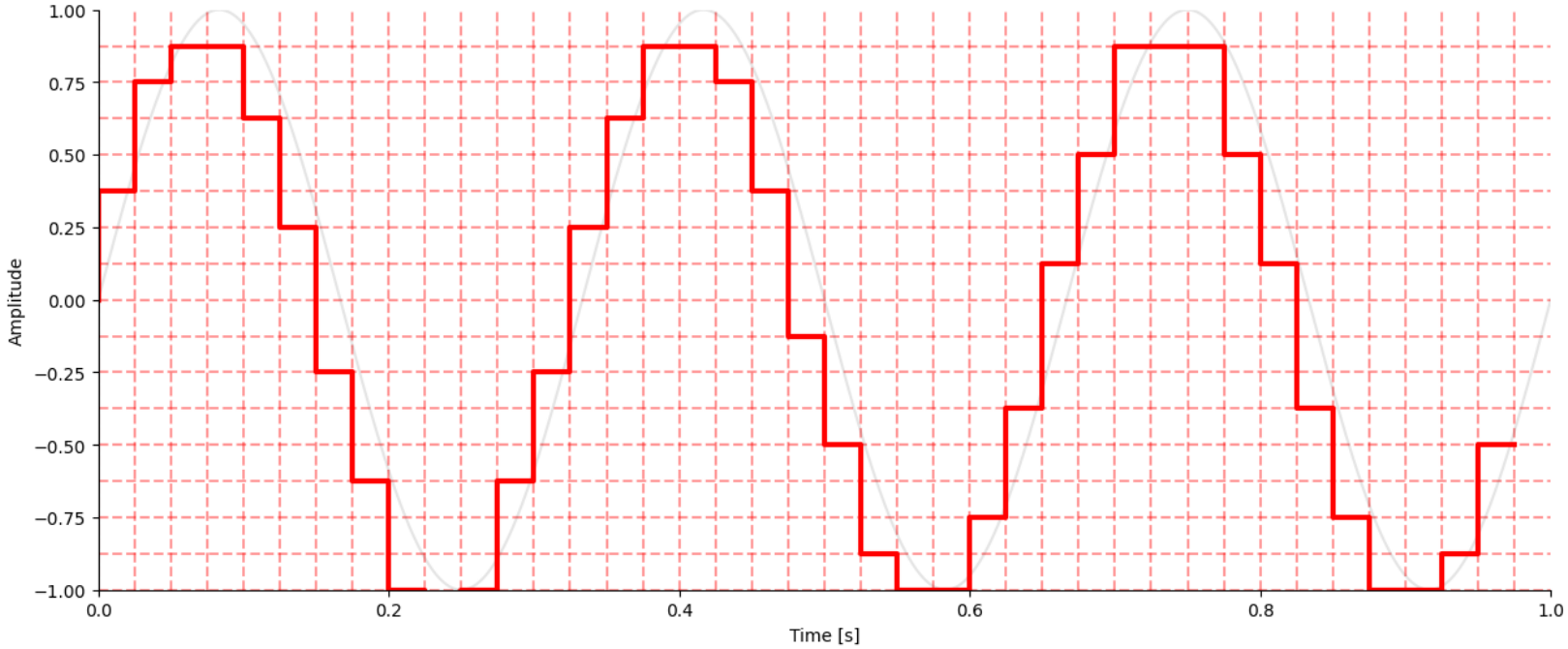
Quantization: Bit Depth



Bit Depth: **4 bits**

DIGITAL SIGNALS

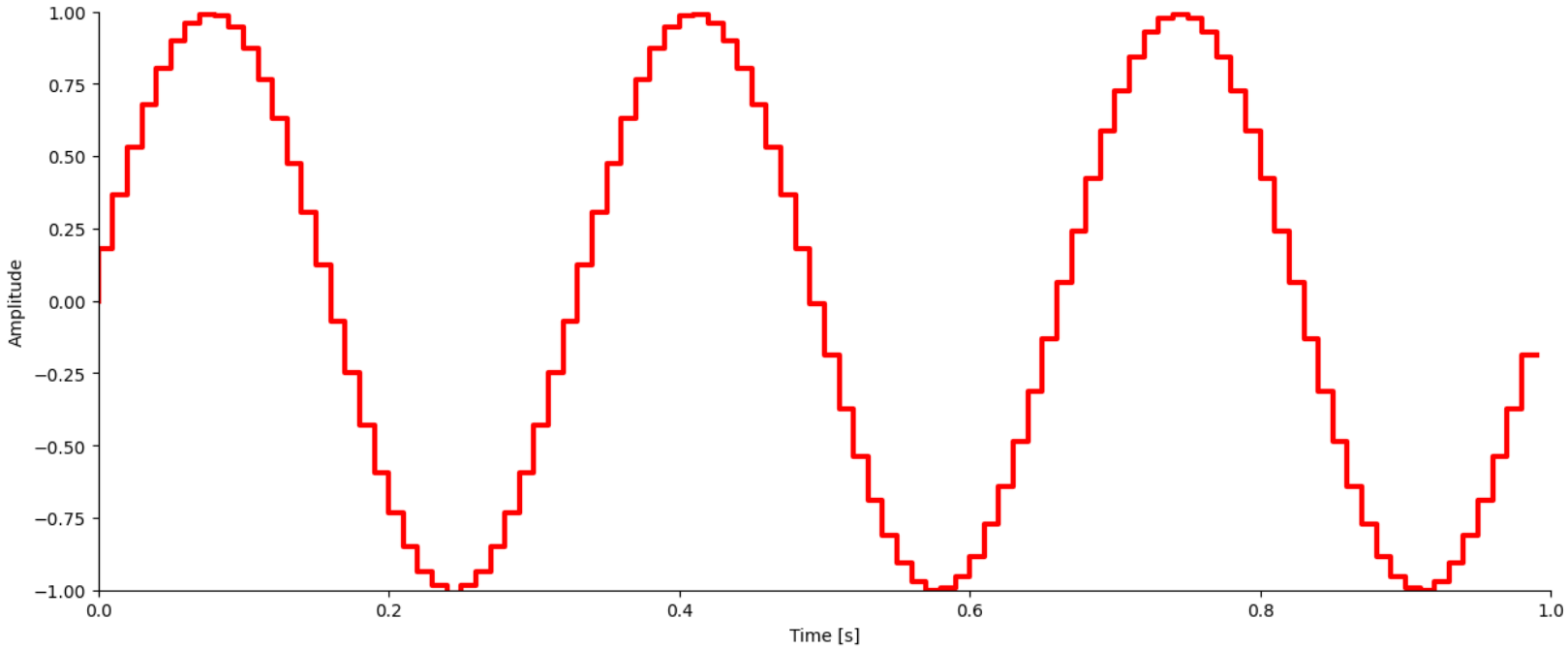
Quantization: Bit Depth



Bit Depth: **4 bits**

DIGITAL SIGNALS

Quantization: Bit Depth

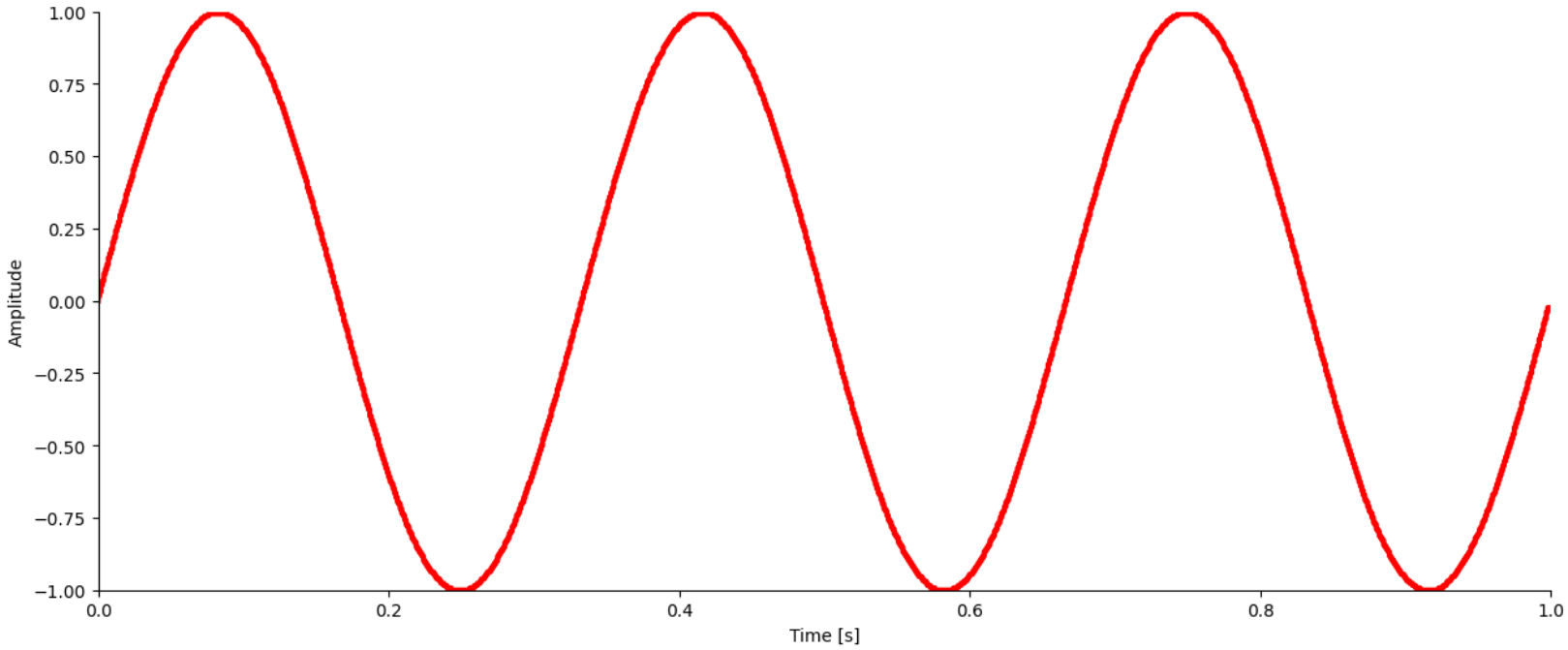


Bit Depth: **8 bits**

Sample Rate: **100 Hz**

DIGITAL SIGNALS

Quantization: Bit Depth

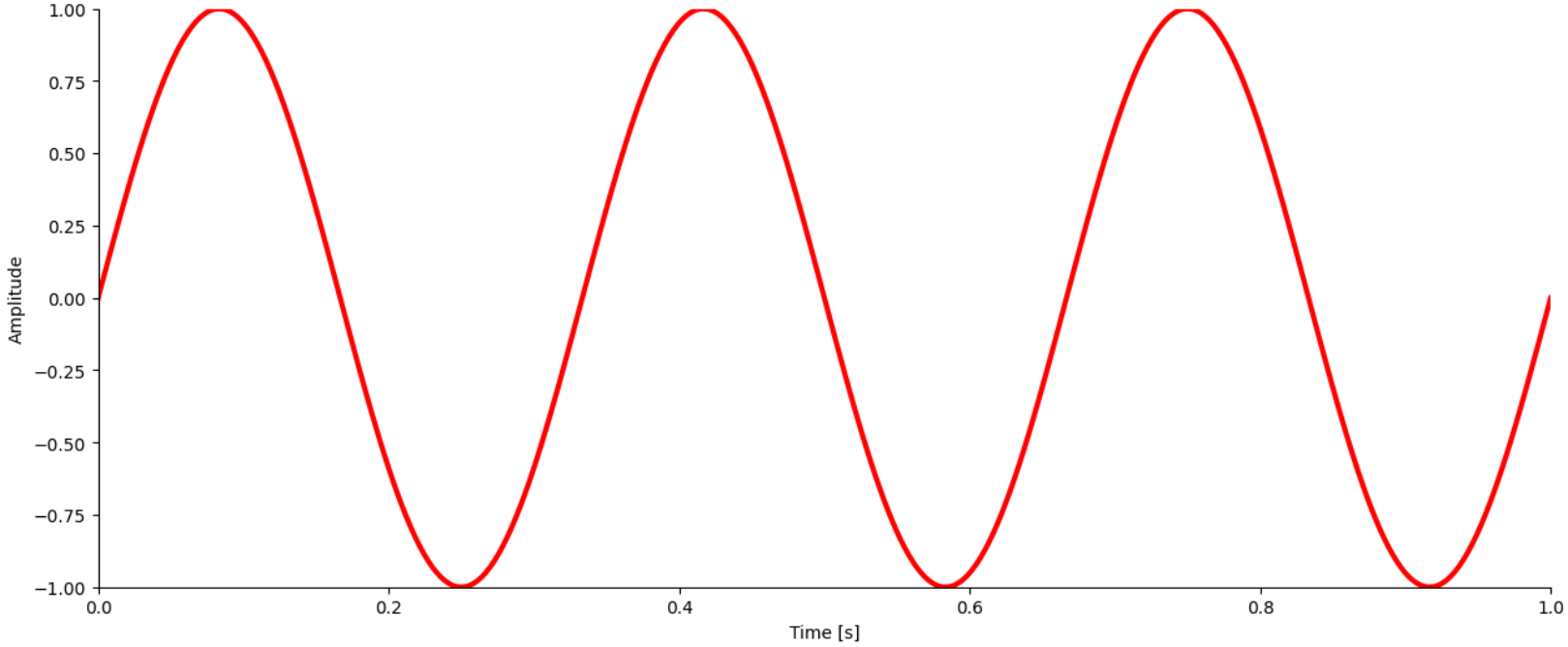


Bit Depth: **8 bits**

Sample Rate: **1000 Hz**

DIGITAL SIGNALS

Quantization: Bit Depth

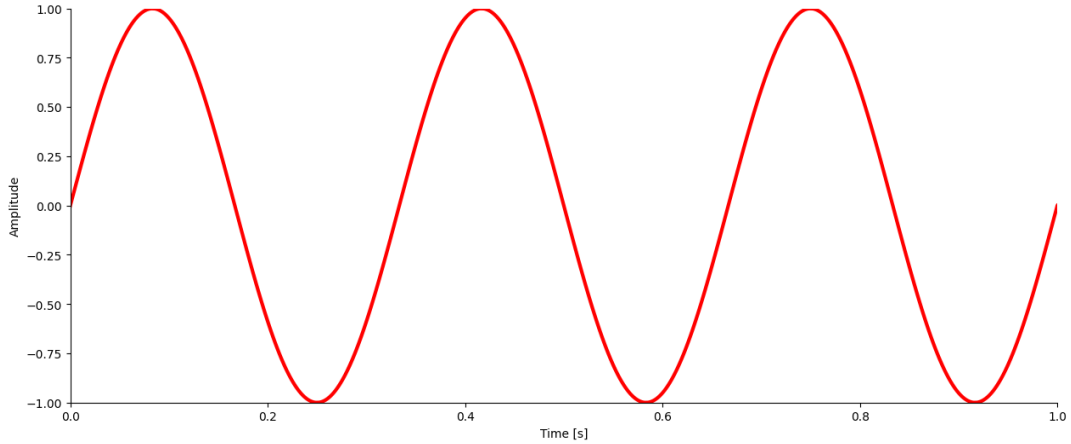


Bit Depth: **16 bits**

Sample Rate: **44100 Hz**

DIGITAL SIGNALS

Quantization: Bit Depth



Typical Bit Depths

16-bit int	**24-bit int**	**32-bit float**
2^{16} values	2^{24} values	2^{32} values

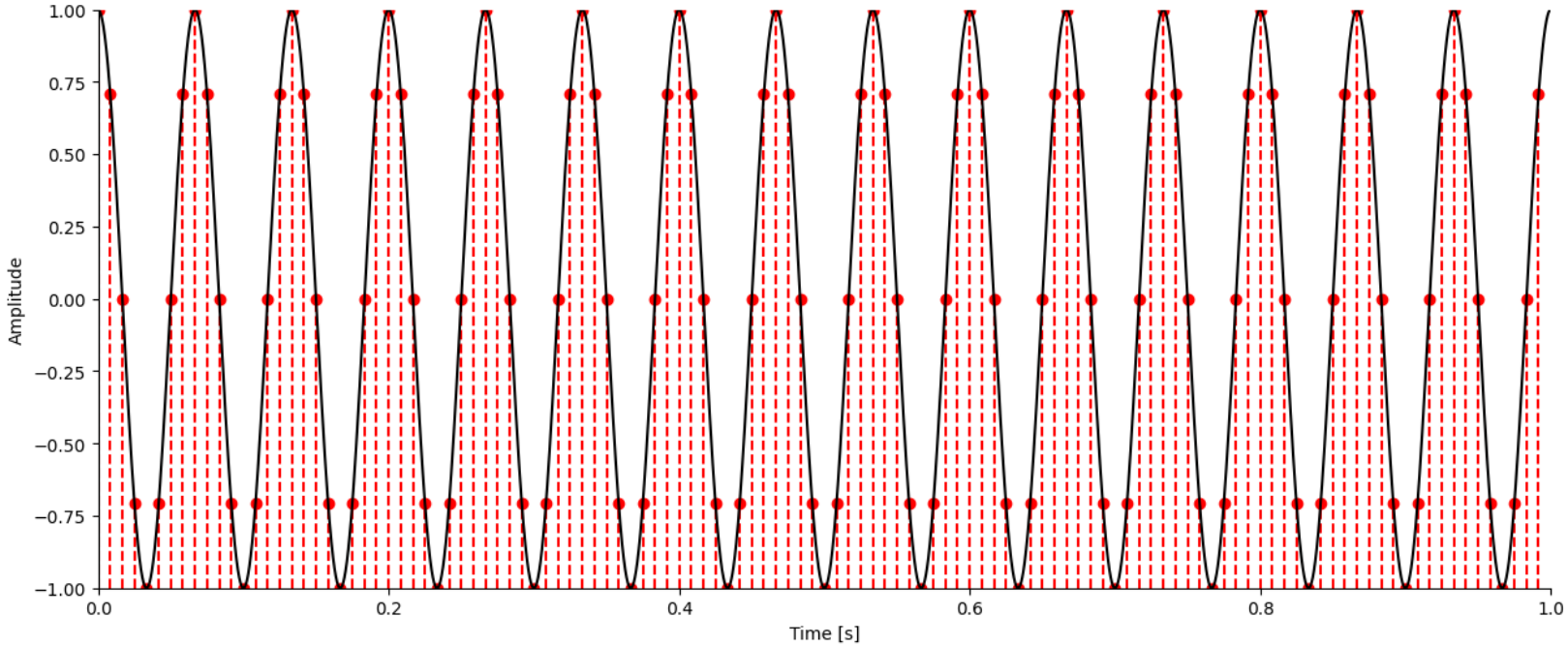
Typical Sampling Rates

44100 Hz	96000 Hz
48000 Hz	192000 Hz

SAMPLING THEORY

Nyquist-Shannon Theorem

Highest Frequency ≤ **Sampling Rate / 2**



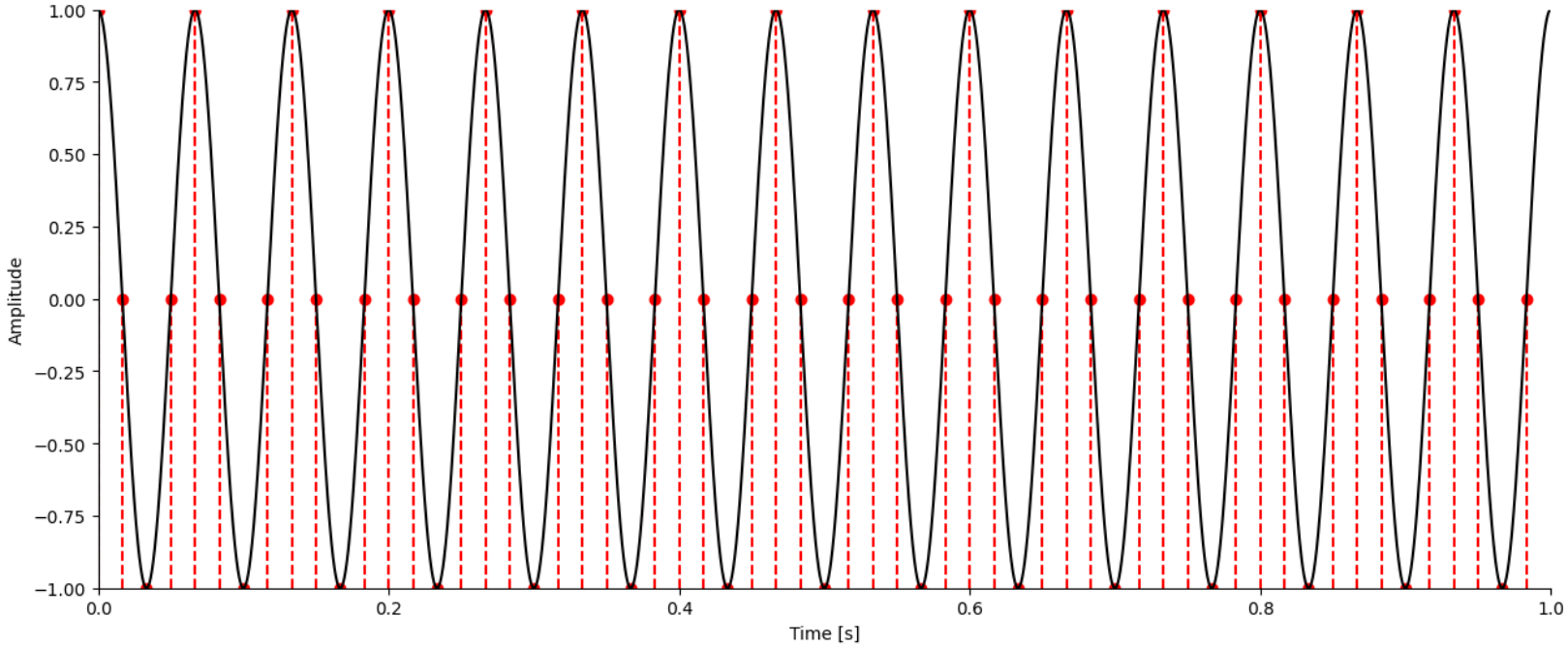
Frequency: 15 Hz

Sample Rate: 120 Hz

SAMPLING THEORY

Nyquist-Shannon Theorem

Highest Frequency ≤ **Sampling Rate / 2**



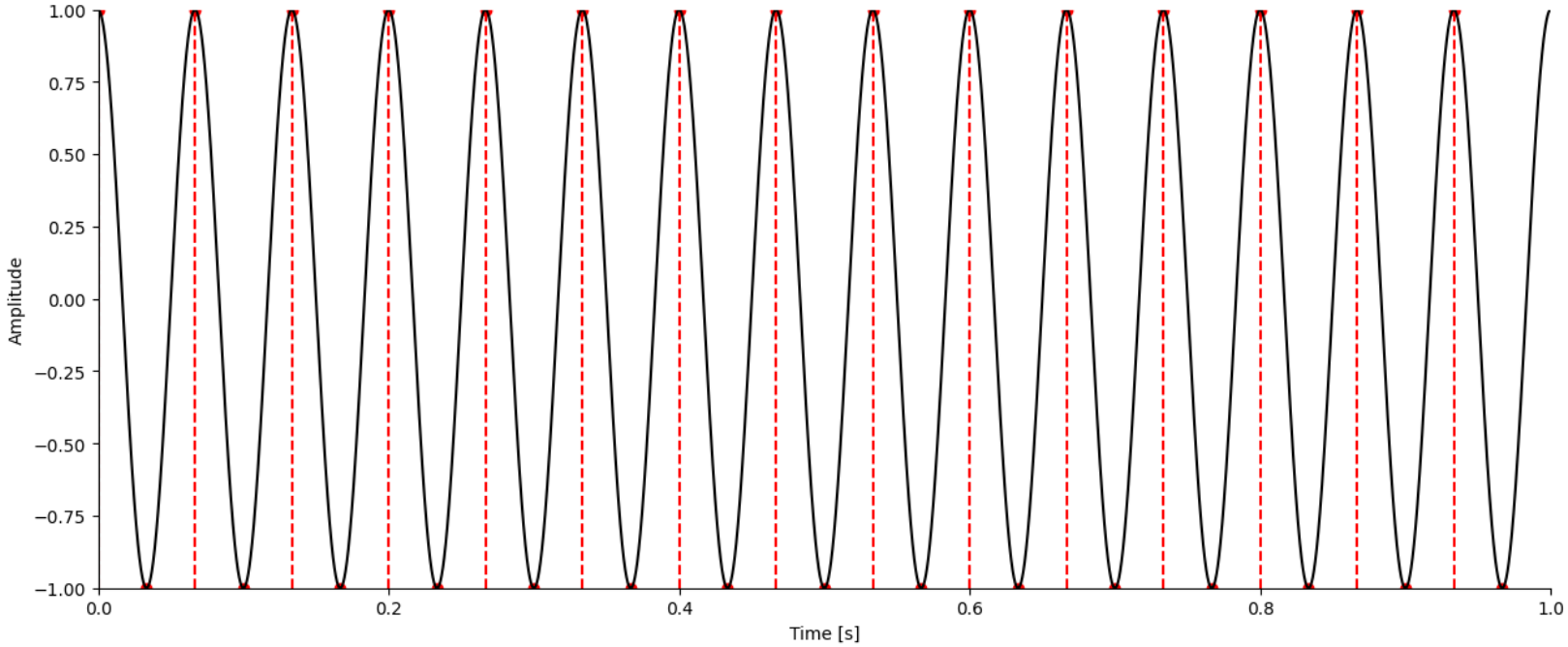
Frequency: 15 Hz

Sample Rate: 60 Hz

SAMPLING THEORY

Nyquist-Shannon Theorem

Highest Frequency ≤ **Sampling Rate / 2**



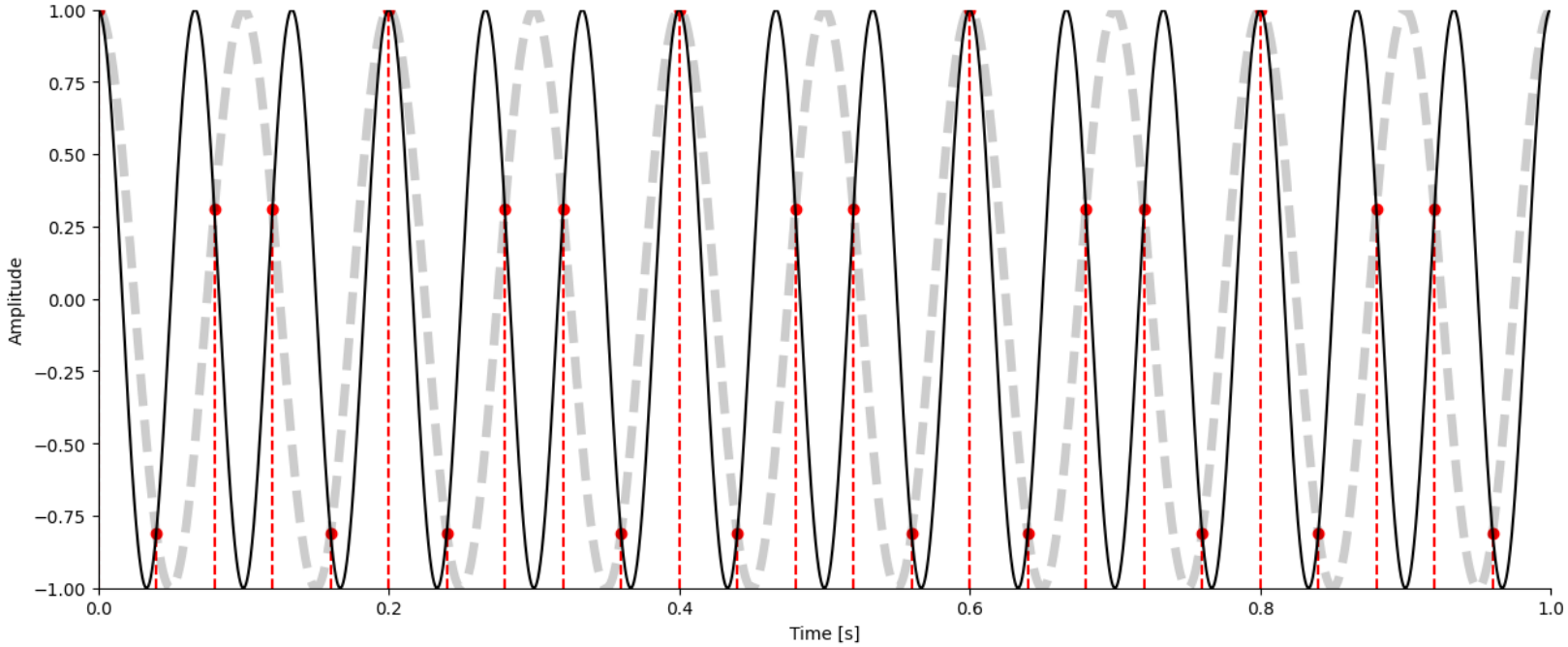
Frequency: 15 Hz

Sample Rate: 30 Hz

SAMPLING THEORY

Nyquist-Shannon Theorem

Highest Frequency ≤ **Sampling Rate / 2**



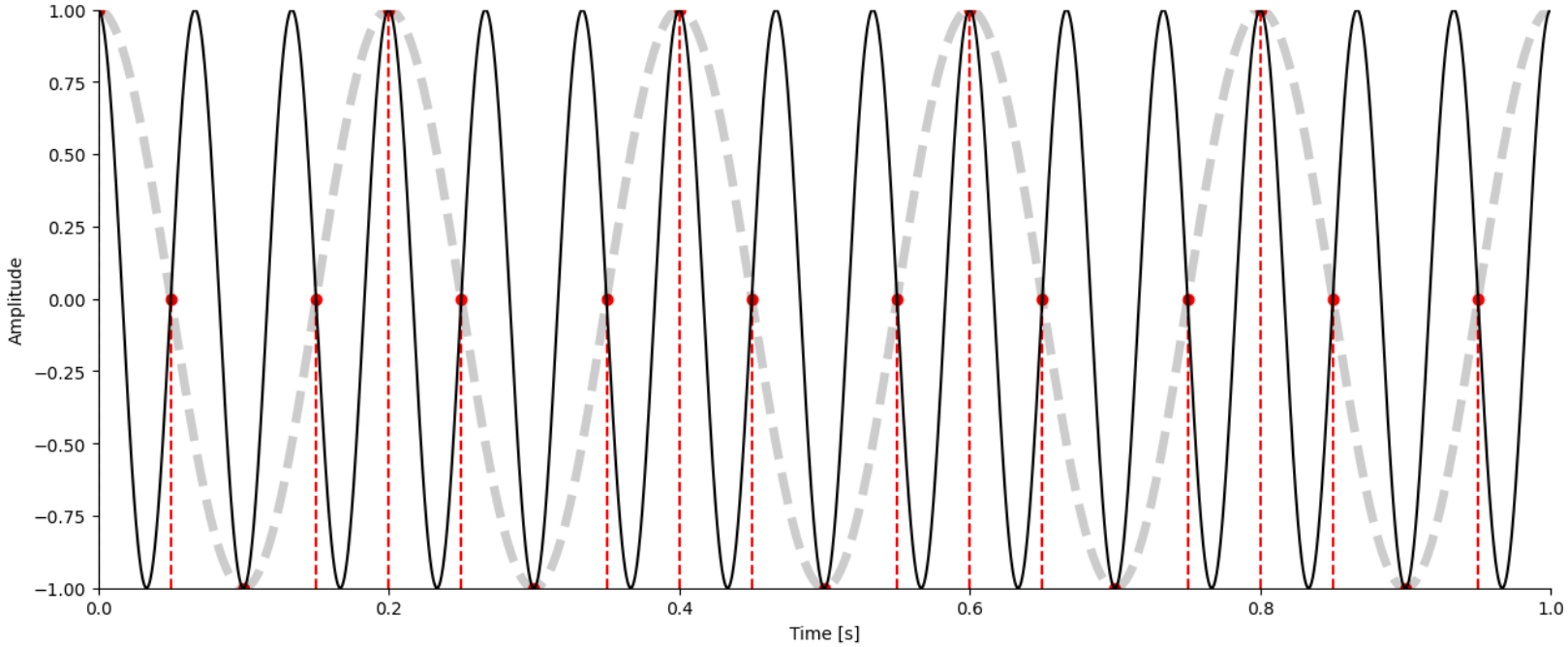
Frequency: 15 Hz

Sample Rate: 25 Hz

SAMPLING THEORY

Nyquist-Shannon Theorem

Highest Frequency ≤ **Sampling Rate / 2**



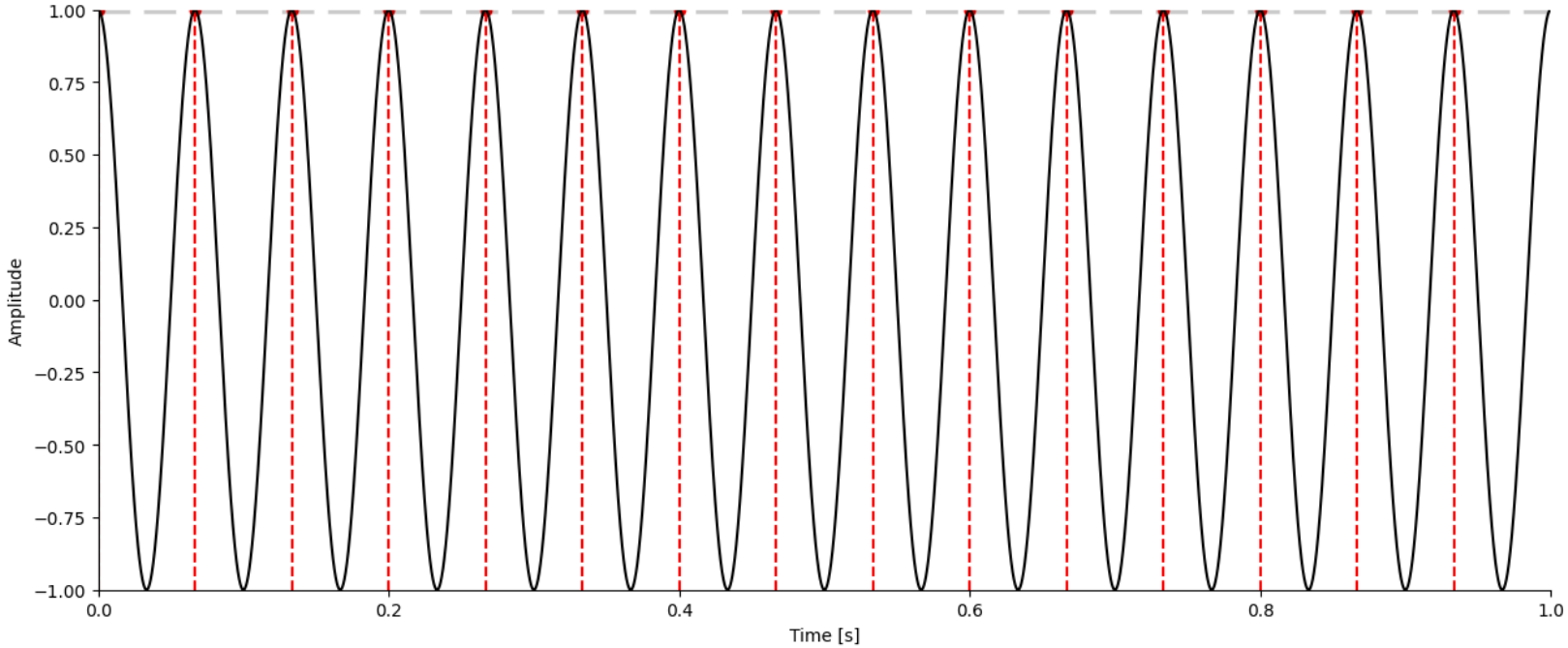
Frequency: 15 Hz

Sample Rate: 20 Hz

SAMPLING THEORY

Nyquist-Shannon Theorem

Highest Frequency ≤ **Sampling Rate / 2**



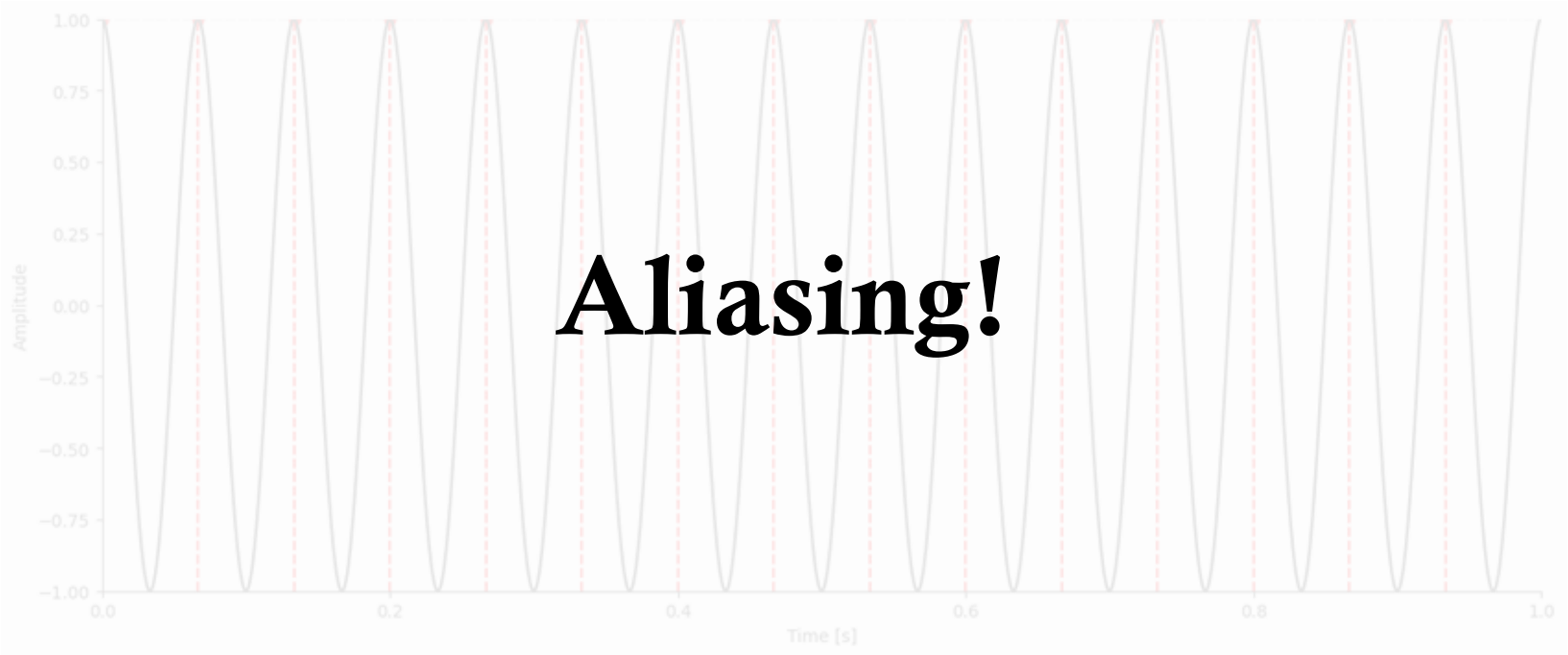
Frequency: 15 Hz

Sample Rate: 15 Hz

SAMPLING THEORY

Nyquist-Shannon Theorem

Highest Frequency ≤ **Sampling Rate / 2**



Aliasing!

Frequency: 15 Hz

Sample Rate: 15 Hz

FOURIER TRANSFORM

Introduction

Decomposition of audio to its constituent frequencies

Consider the signal

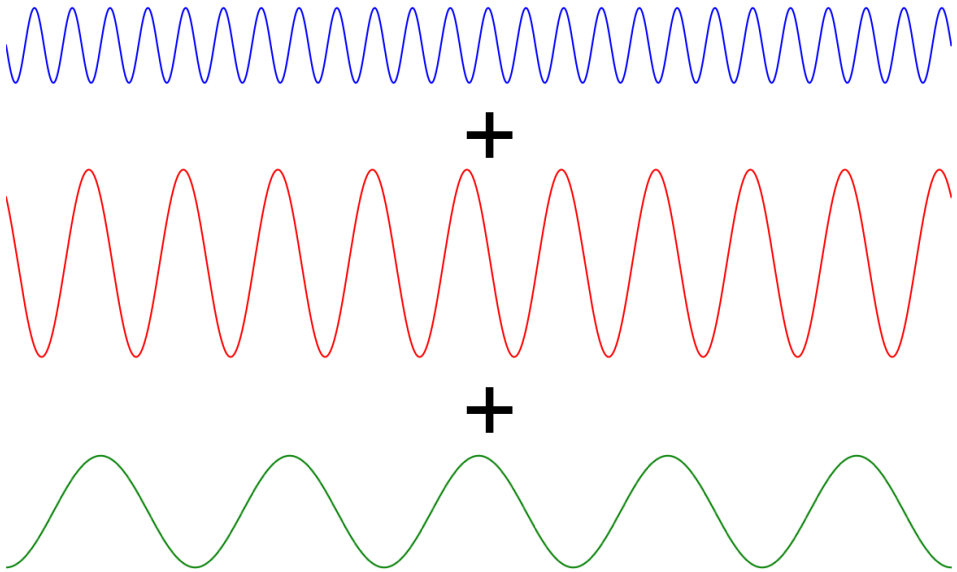


FOURIER TRANSFORM

Introduction

Decomposition of audio to its constituent frequencies

Decomposed to sines



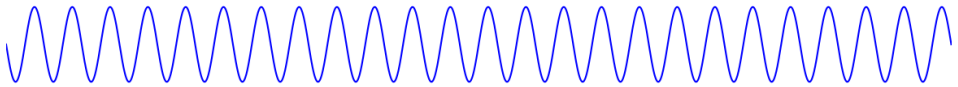
FOURIER TRANSFORM

Introduction

Decomposition of audio to its constituent frequencies

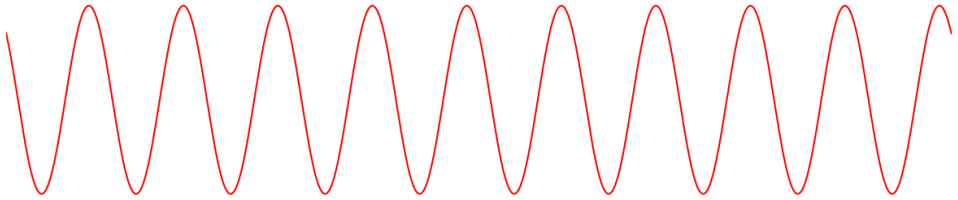
Decomposed to sines

$f_1 = 5$ Hz



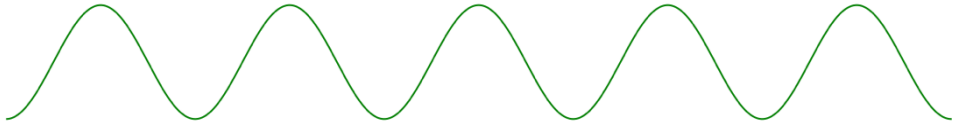
+

$f_2 = 2$ Hz



+

$f_3 = 1$ Hz



$$\mathbf{0.2}\cos\left(2\pi f_1 t + \frac{\pi}{\mathbf{2}}\right)$$

$$\mathbf{0.5}\cos\left(2\pi f_2 t + \frac{\pi}{\mathbf{4}}\right)$$

$$\mathbf{0.3}\cos(2\pi f_3 t - \pi)$$

FOURIER TRANSFORM

Introduction

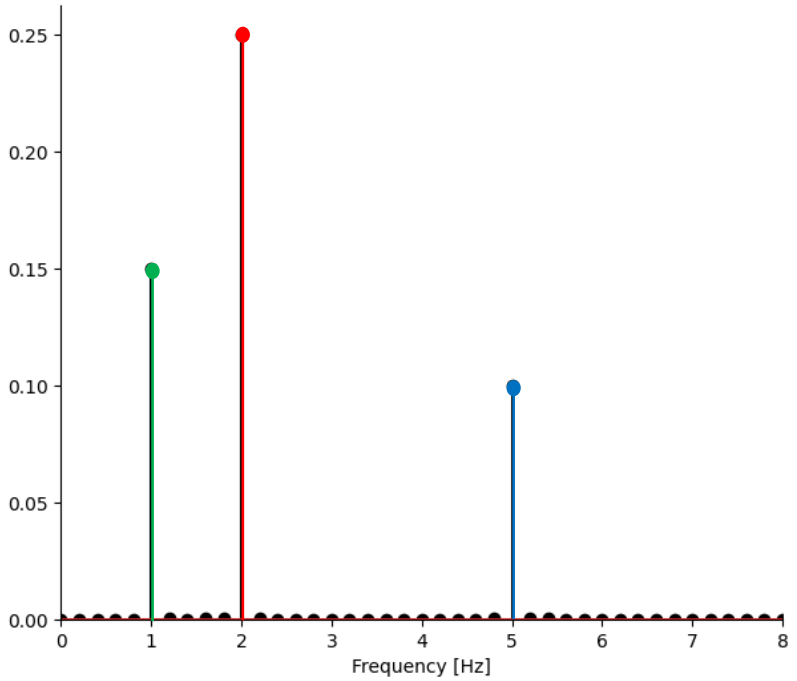
Decomposition of audio to its constituent frequencies

$$0.2\cos\left(2\pi f_1 t + \frac{\pi}{2}\right)$$

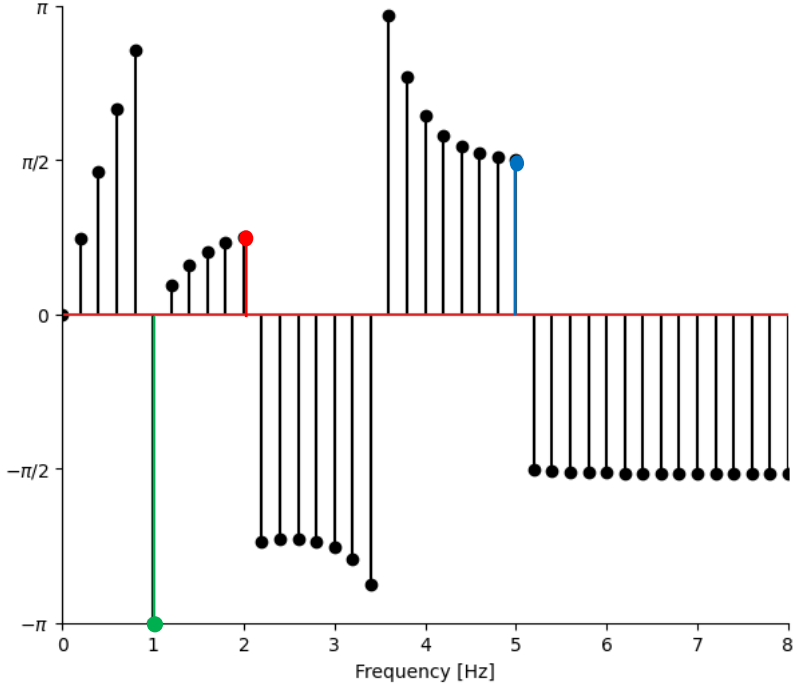
$$0.5\cos\left(2\pi f_2 t + \frac{\pi}{4}\right)$$

$$0.3\cos(2\pi f_3 t - \pi)$$

Magnitude Response



Phase Response



FOURIER TRANSFORM

Applications

Decomposition of audio to its constituent frequencies

Applications

- **Filtering**
- **Spectral Analysis**
- **Fast Convolution**
- **Audio Effects**

TABLE OF CONTENTS



AUDIO FUNDAMENTALS



PYTHON LIBRARIES FOR AUDIO



AUDIO CODECS



PYTHON LIBRARIES FOR AUDIO

PYTHON AUDIO LIBRARIES

Librosa

Installation `pip install librosa`

Common Use Cases

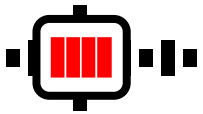
Audio Content Analysis



Audio Visualization



Feature Extraction



Audio Effects

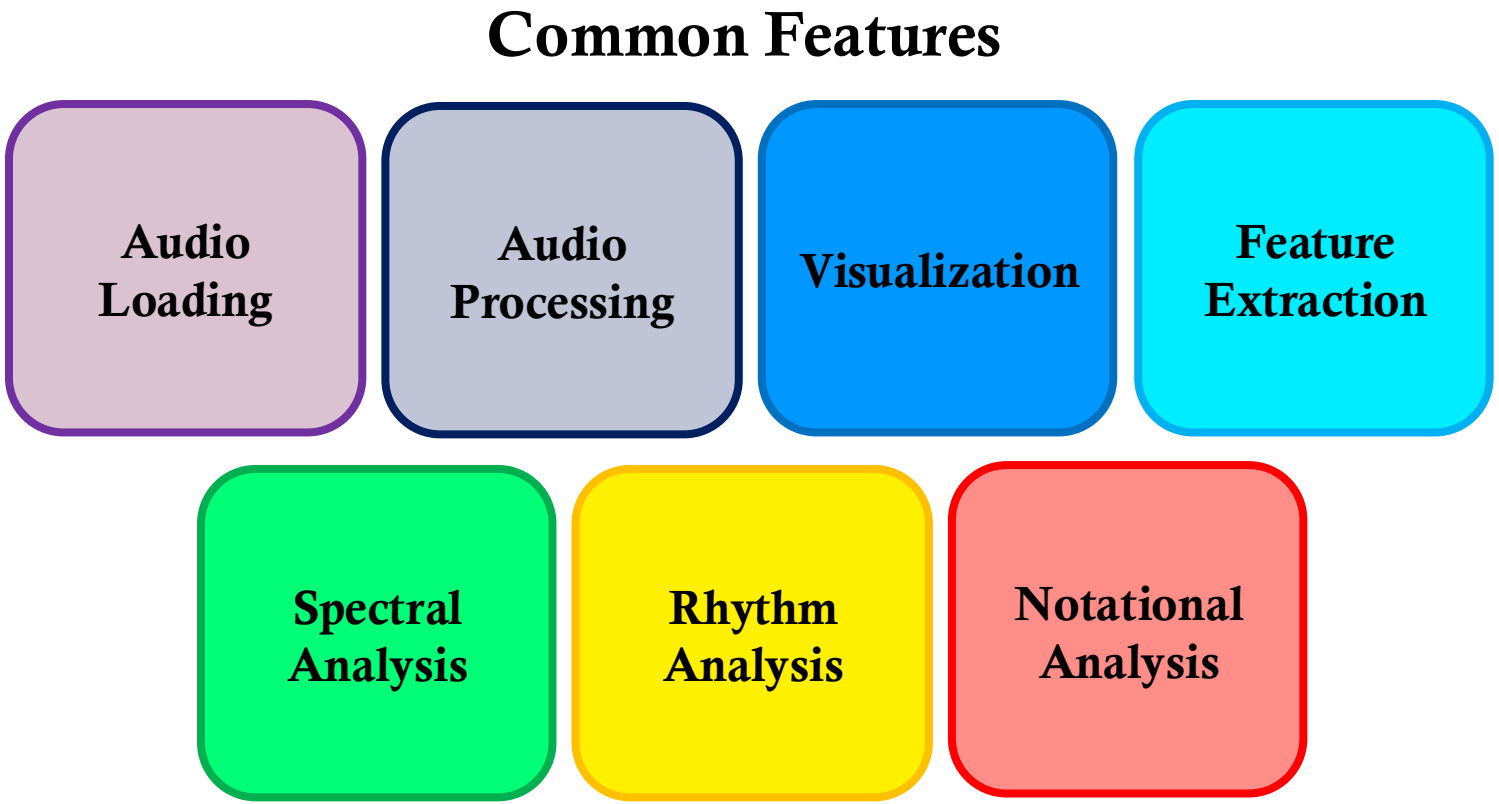


and many more…*

*https://librosa.org/doc/latest/api/index.html

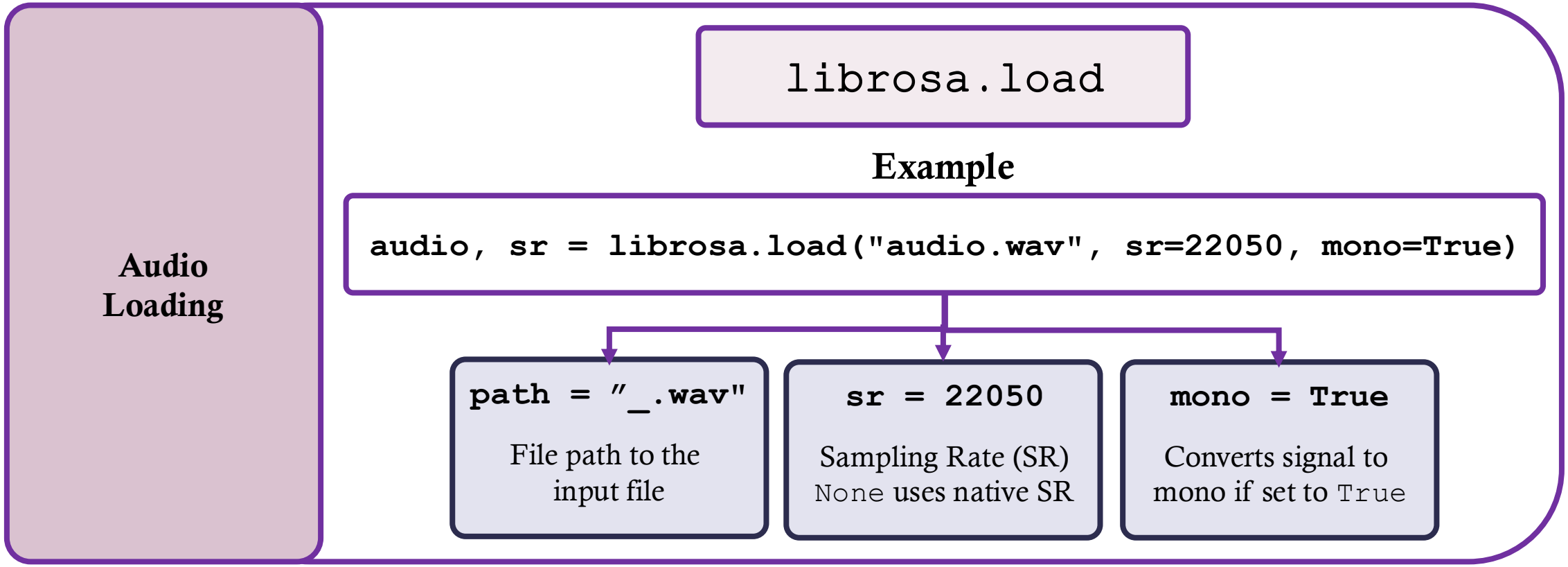
PYTHON AUDIO LIBRARIES

Librosa: Features



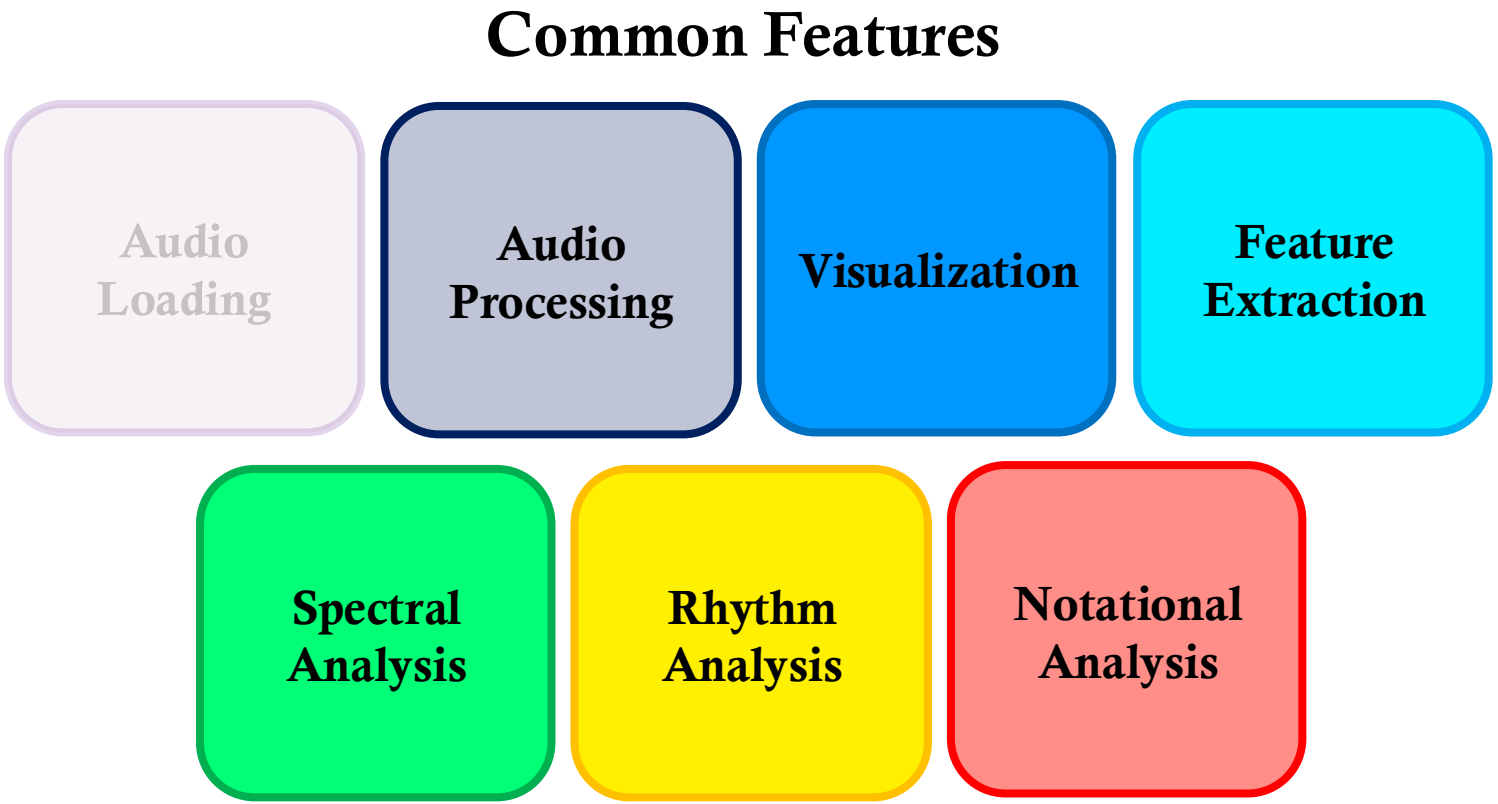
PYTHON AUDIO LIBRARIES

Librosa: Features



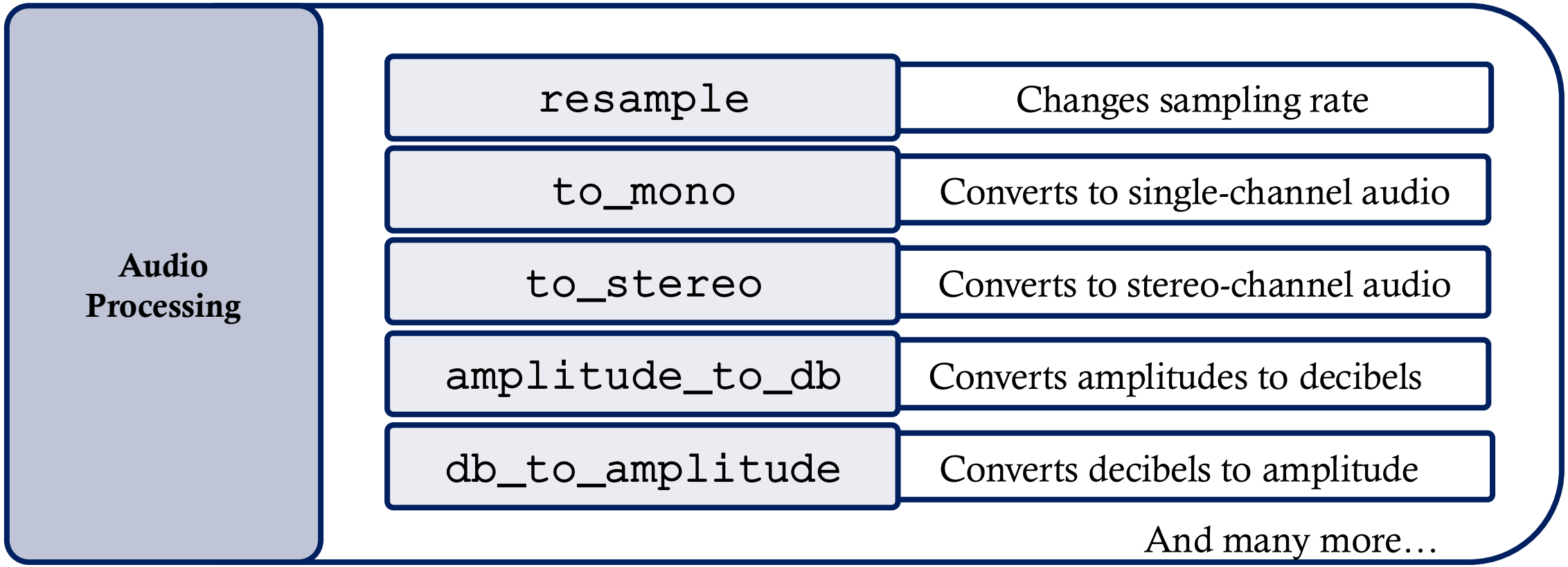
PYTHON AUDIO LIBRARIES

Librosa: Features



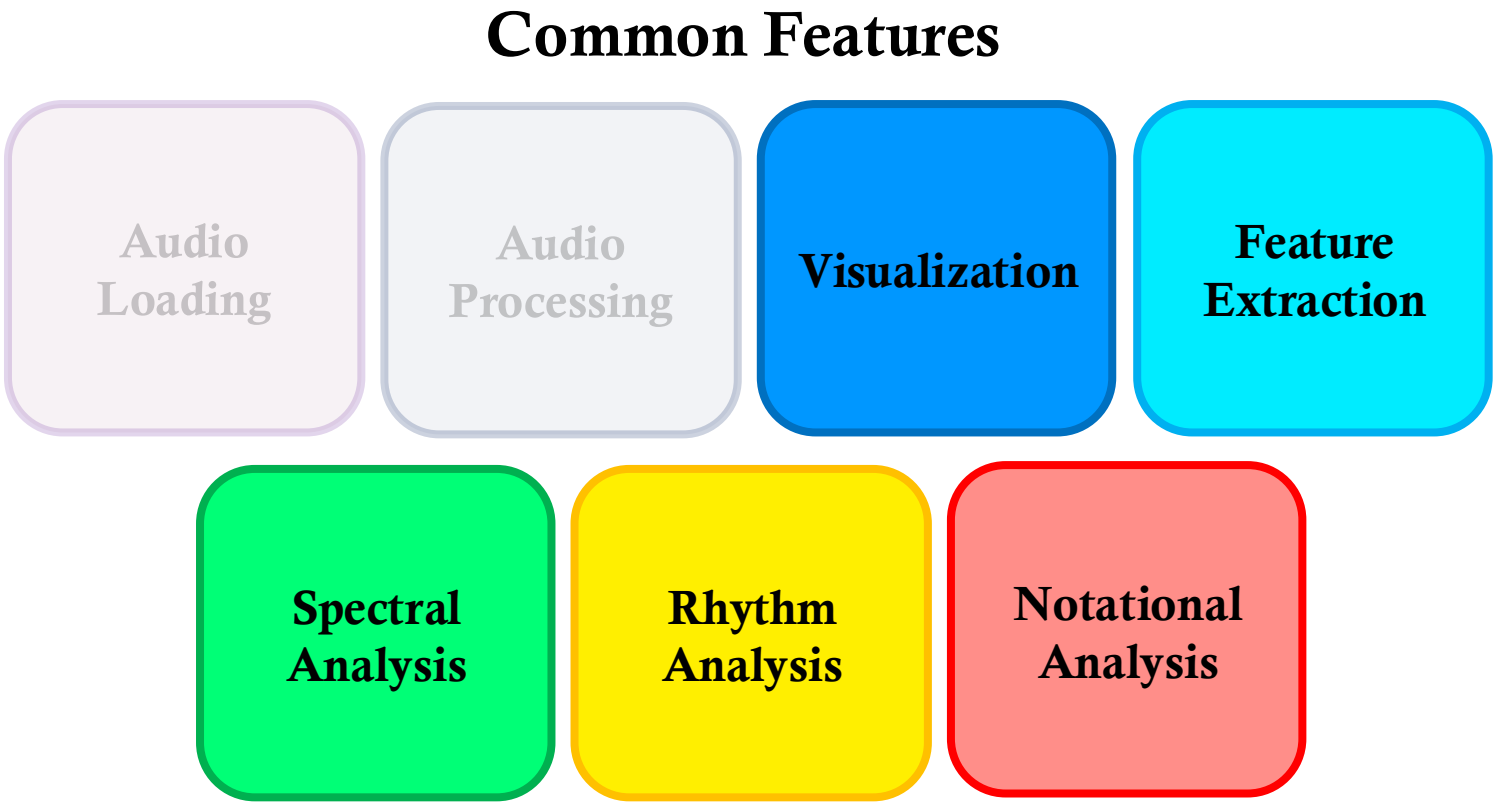
PYTHON AUDIO LIBRARIES

Librosa: Features



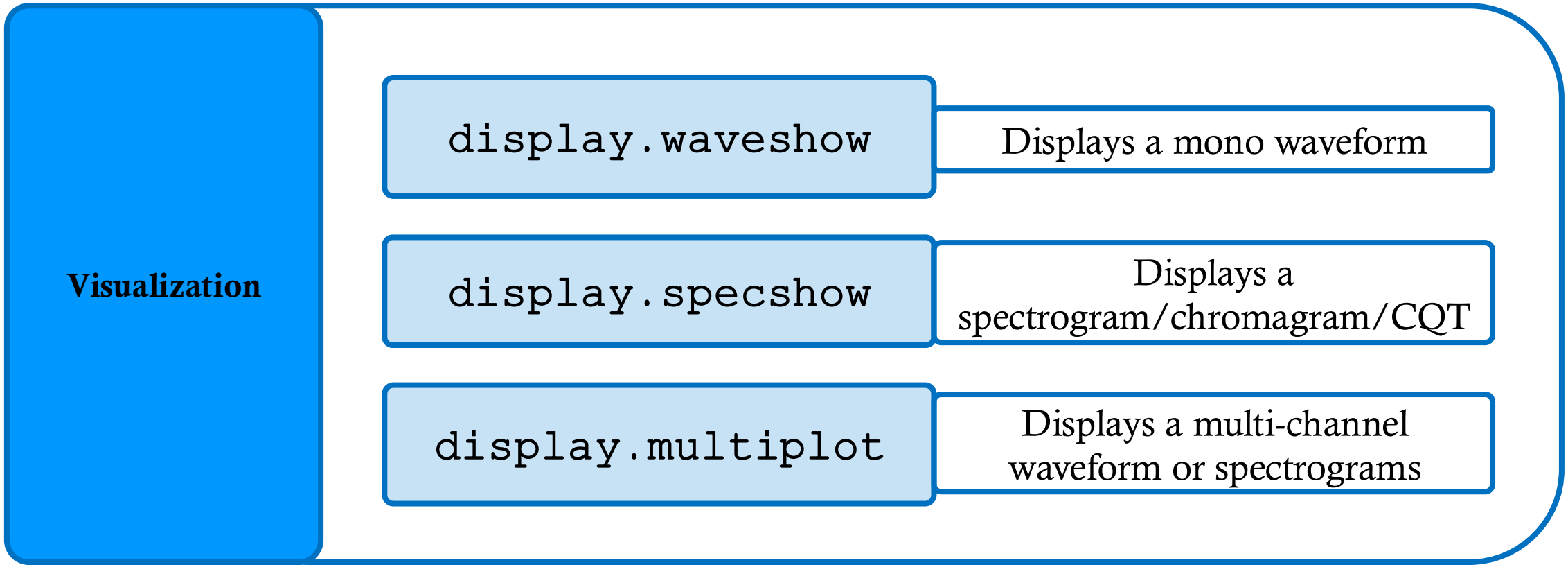
PYTHON AUDIO LIBRARIES

Librosa: Features



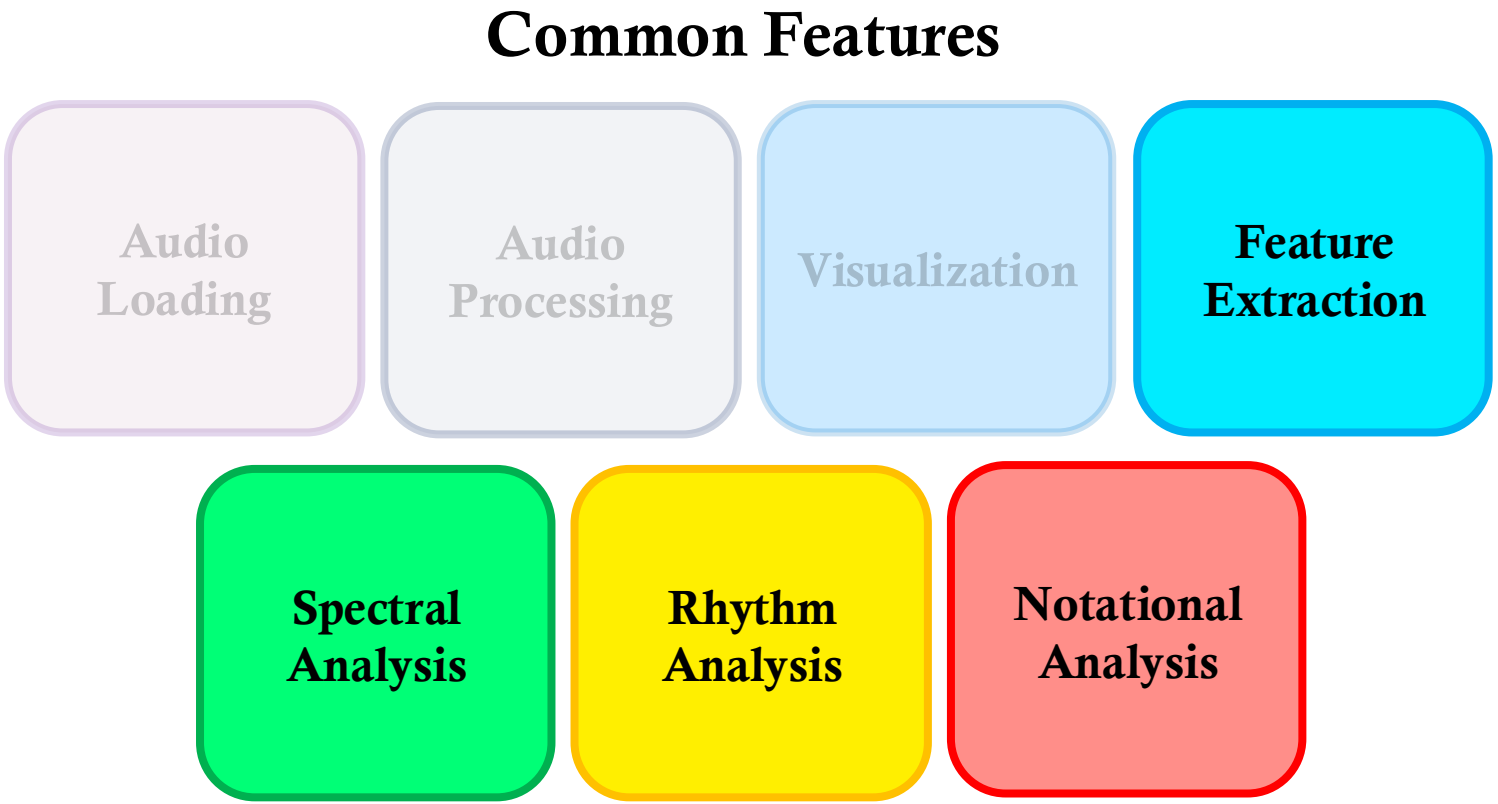
PYTHON AUDIO LIBRARIES

Librosa: Features



PYTHON AUDIO LIBRARIES

Librosa: Features



PYTHON AUDIO LIBRARIES

Librosa: Features

Feature Extraction

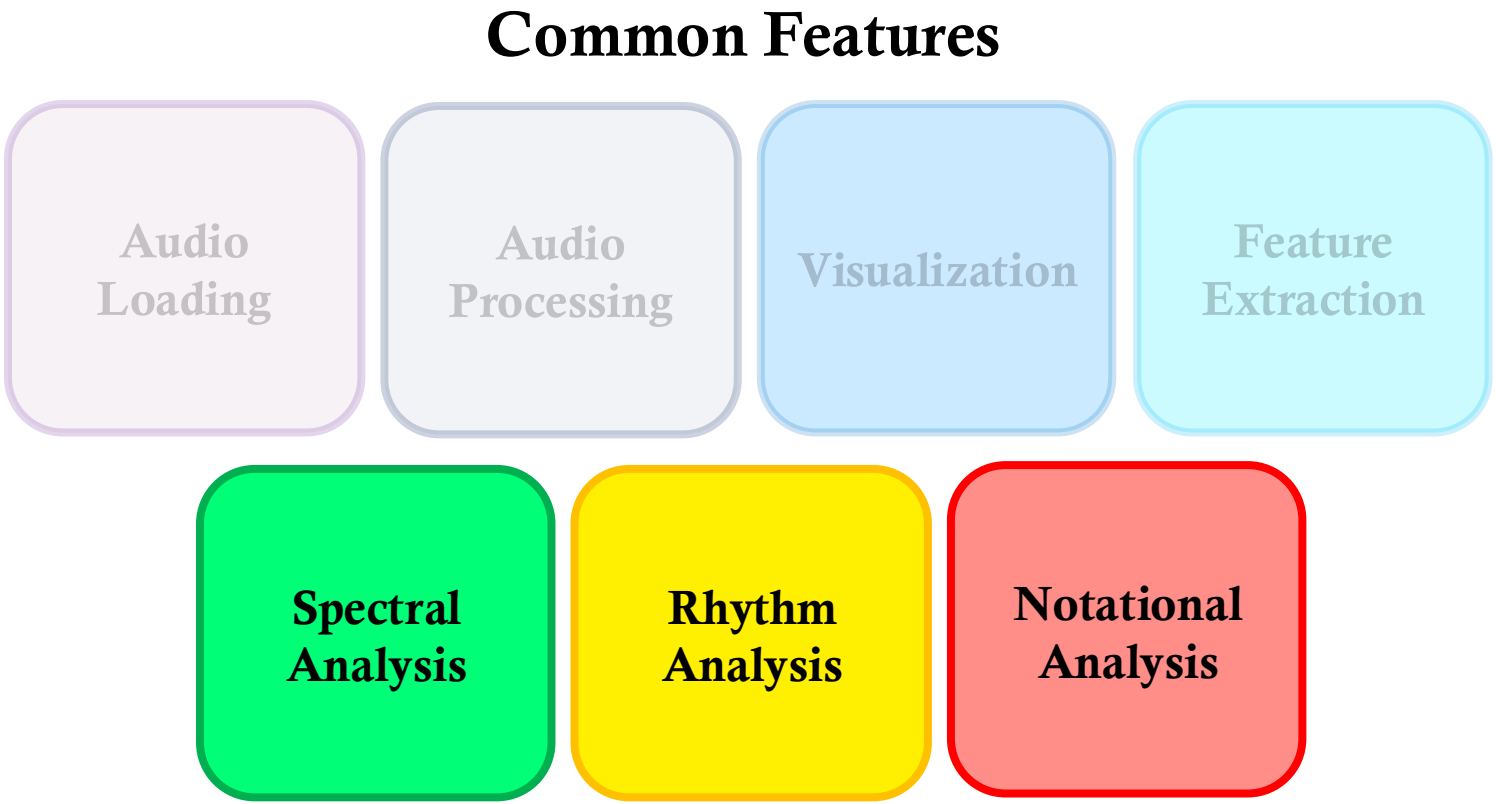
`feature.chroma_cqt`	Constant Q-Chromagram
`feature.melspectrogram`	Computes Mel-scale Spectrogram
`feature.mfcc`	Mel-Frequency Cepstral Coefficients
`feature.rms`	Frame-wise root mean squared
`feature.spectral_centroid`	Frame-wise spectral centroid

And many more…

https://librosa.org/doc/latest/api/feature.html

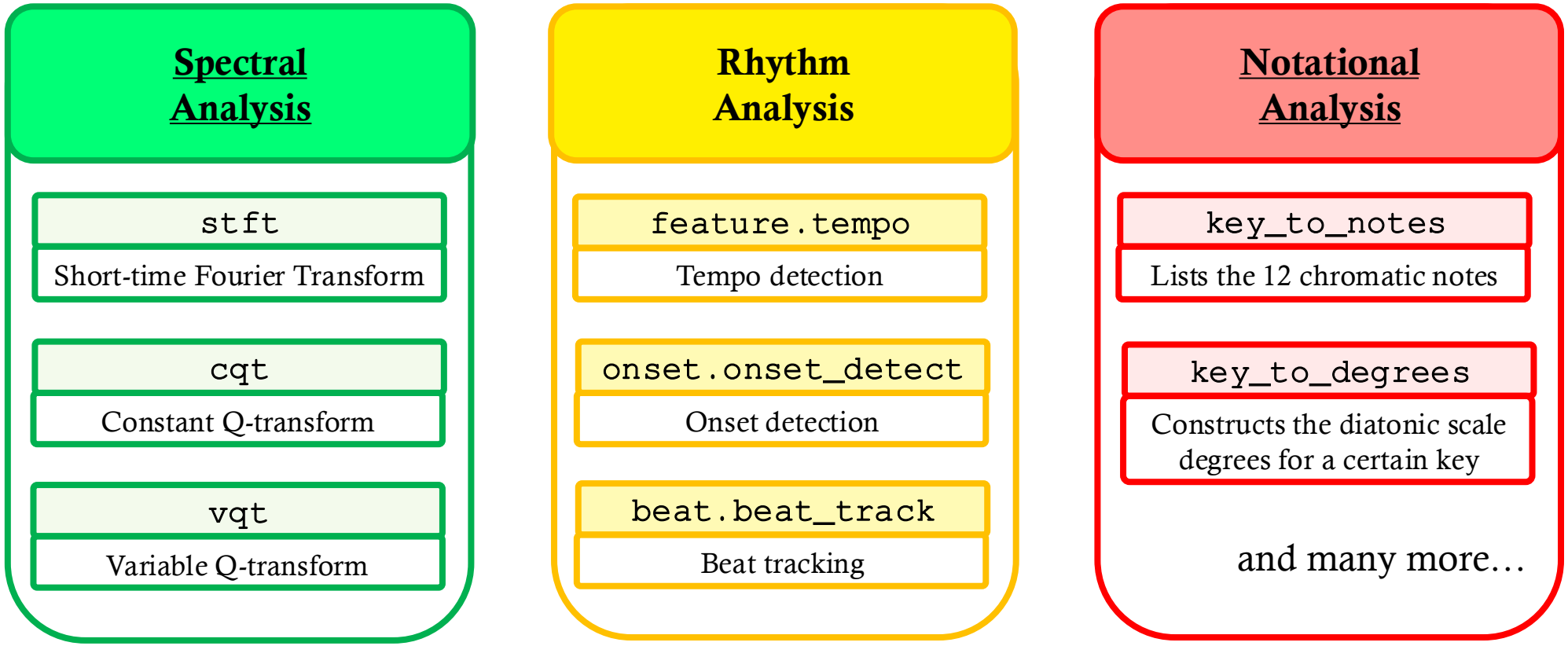
PYTHON AUDIO LIBRARIES

Librosa: Features



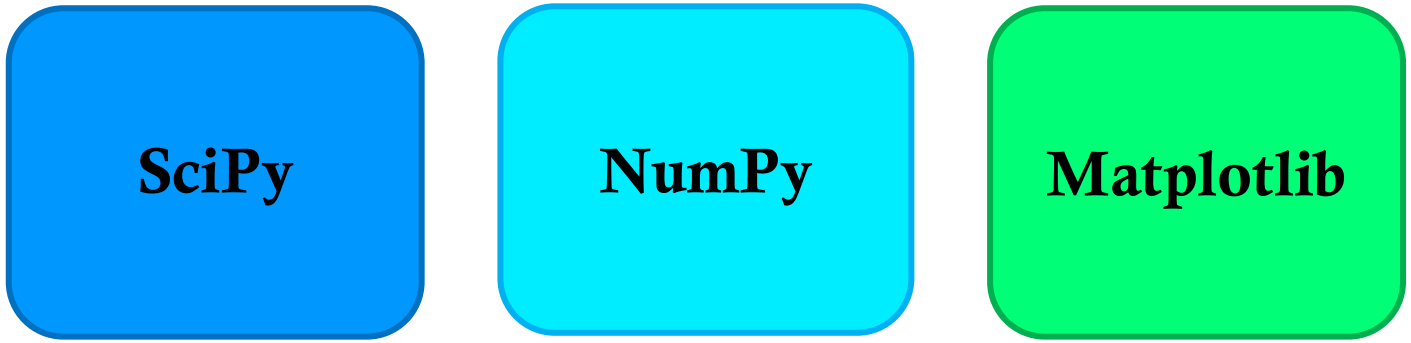
PYTHON AUDIO LIBRARIES

Librosa: Features



PYTHON AUDIO LIBRARIES

Other Libraries



PYTHON AUDIO LIBRARIES

Other Libraries

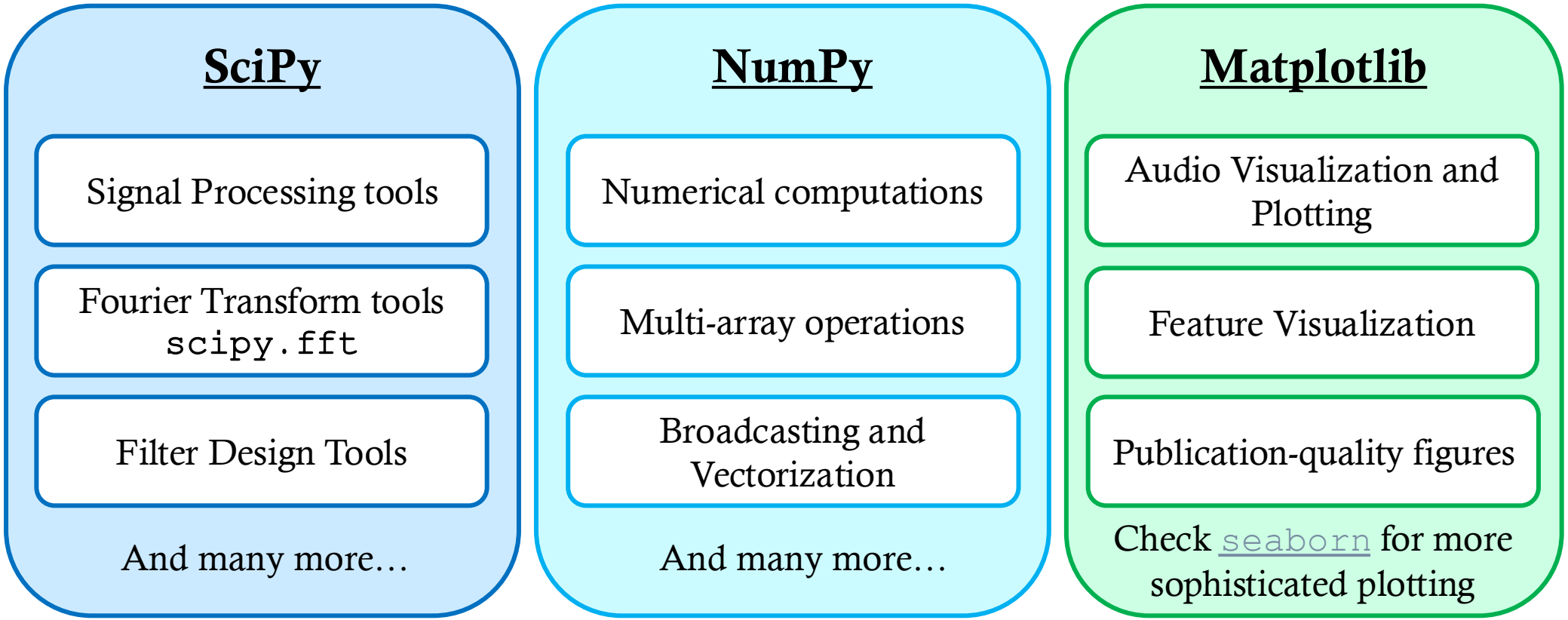


TABLE OF CONTENTS



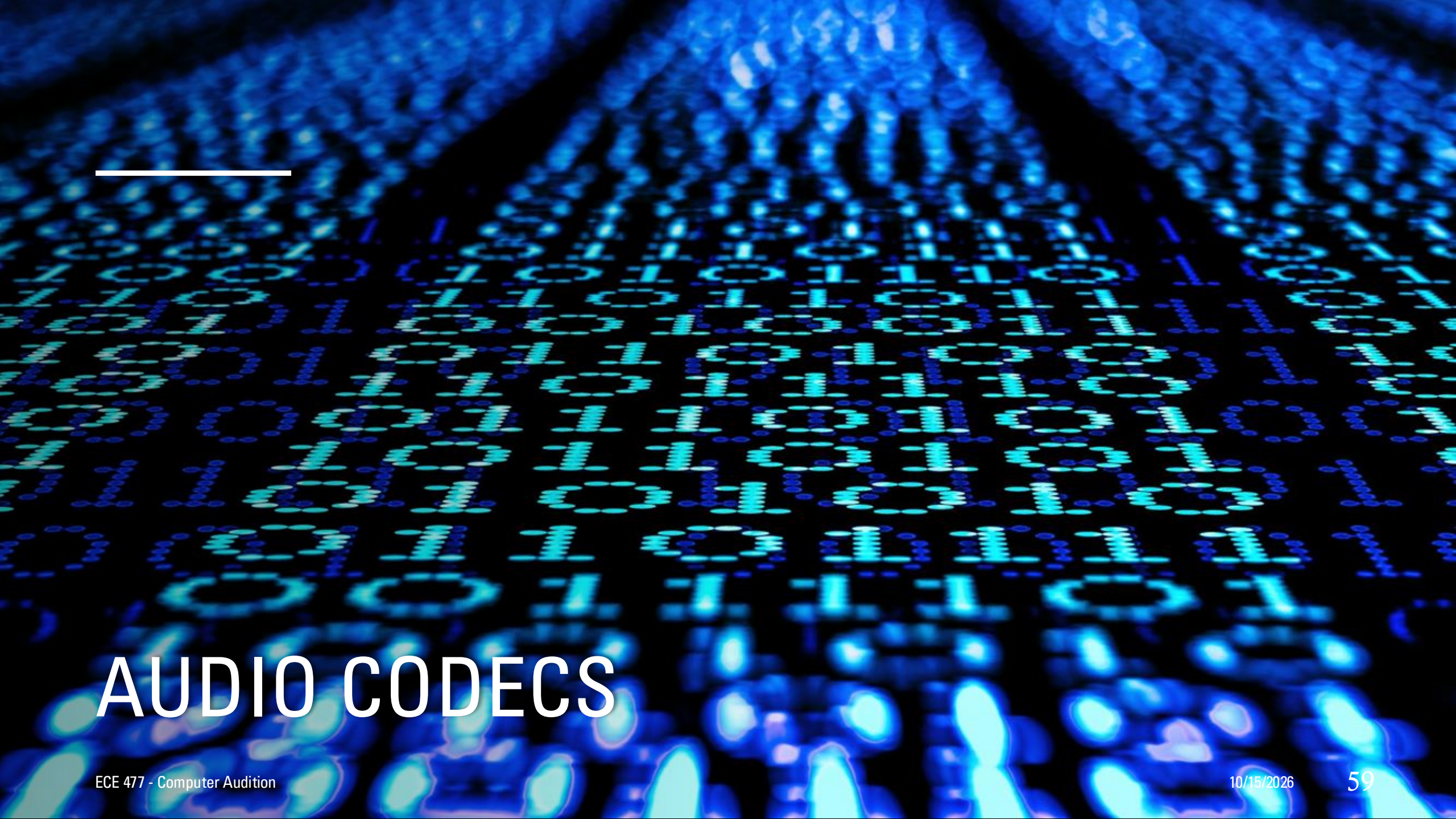
AUDIO FUNDAMENTALS



PYTHON LIBRARIES FOR AUDIO



AUDIO CODECS

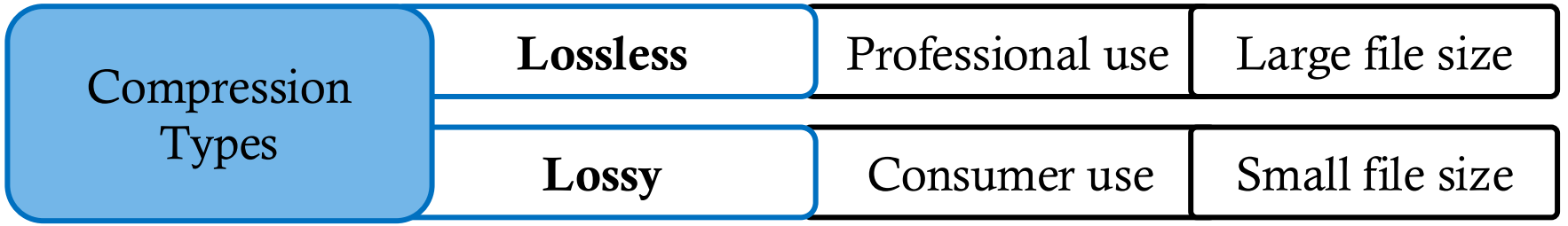


AUDIO CODECS

AUDIO CODECS

Introduction

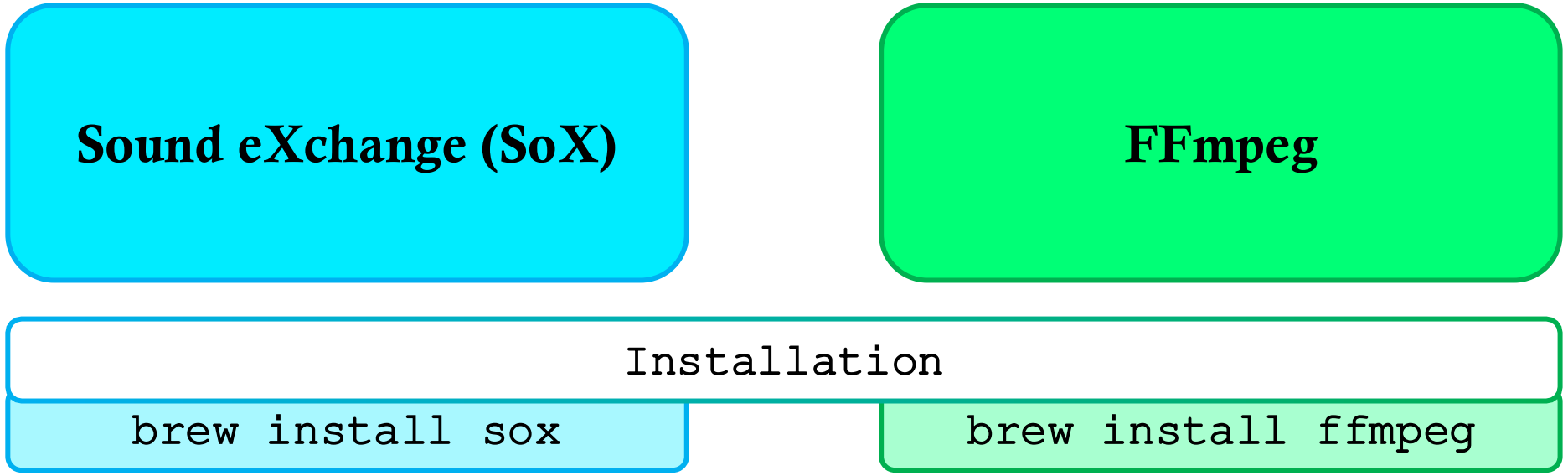
A program that encodes and decodes audio to reduce file size while maintaining quality



Format	Type	Quality	Size
WAV	Lossless	Perfect	Large
FLAC	Lossless	Perfect	Medium
MP3	Lossy	Good	Small
AAC	Lossy	Better	Smaller

AUDIO CODECS

Libraries



AUDIO CODECS

Libraries

Sound eXchange (SoX)

- Command-line audio processing
- Format conversion
- Audio effects
- Resampling

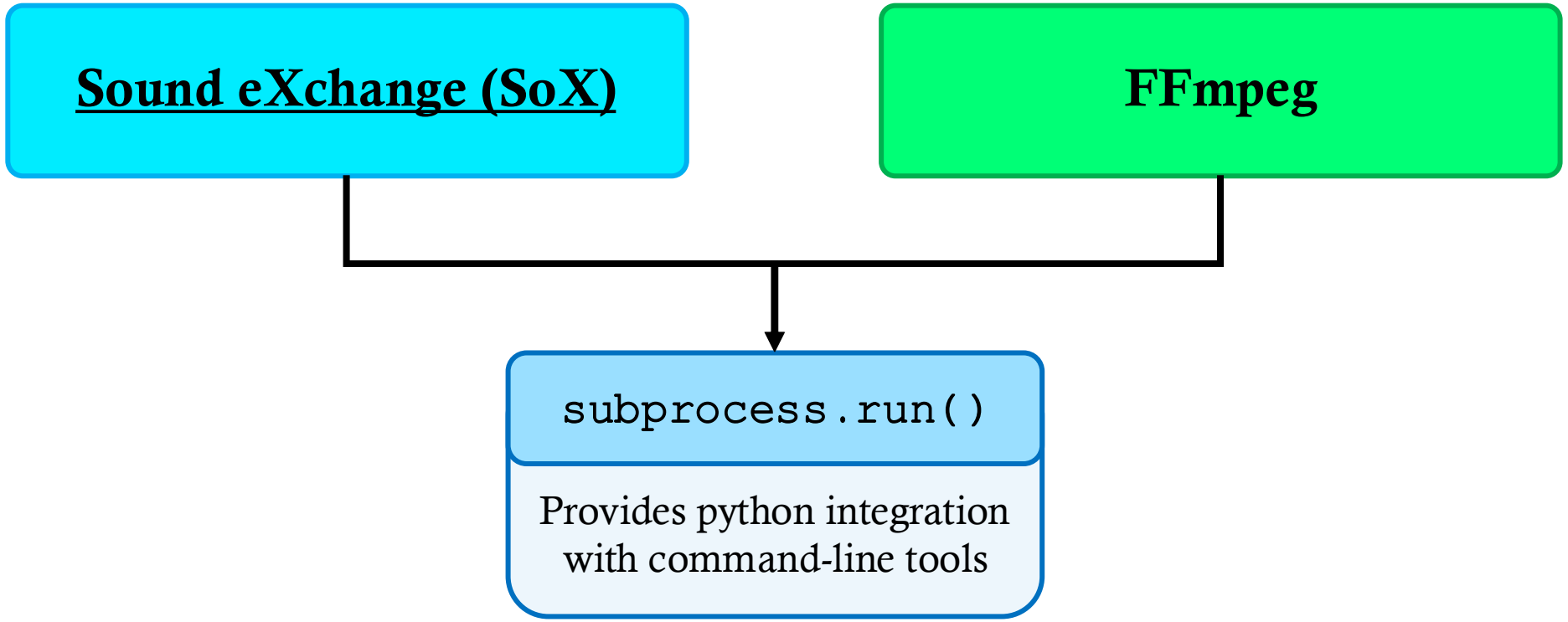
FFmpeg

- Audio-video encoding and decoding
- Format conversion and transcoding
- Filtering
- Metadata manipulation

and many more…

AUDIO CODECS

Libraries: Python Integration



AUDIO CODECS

```python
import subprocess

def convert_with_sox(input_file , output_file , effect=None):
    cmd = ['sox', input_file , output_file]
    if effect:
        cmd.extend(effect.split())
    result = subprocess.run(cmd , capture_output=True , text=True)
    return result.returncode == 0

def extract_audio_ffmpeg(video_file , audio_file):
    cmd = ['ffmpeg', '-i', video_file , '-vn', '-acodec', 'libmp3lame',
           '-ab', '192k', audio_file]
    result = subprocess.run(cmd , capture_output=True , text=True)
    return result.returncode == 0

# Example usage
convert_with_sox('input.wav', 'output.wav', 'reverb 0.5')
extract_audio_ffmpeg('video.mp4', 'audio.mp3')
```

Libraries: Python Integration

TABLE OF CONTENTS



AUDIO FUNDAMENTALS



PYTHON LIBRARIES FOR AUDIO



AUDIO CODECS



SUMMARY

SUMMARY

Audio Fundamentals

- Digital Signals
- Sampling Theory
- Quantization
- Fourier Transform

Python Programming for Audio

- Librosa
- SciPy
- NumPy
- Matplotlib

Audio Codecs

- Introduced Codecs
- SoX and FFmpeg
- Python subprocess

THANK YOU!

Questions?

ECE 477: Computer Audition
University of Rochester - ECE Department

```
python -c ”import librosa; print('Happy audio processing!')”
```