EECS 1710
Programming for Digital Media

Lecture 15 :: Intro to Audio

YORK UNIVERSITÉ UNIVERSITY

Useful Types & where to find them..

Built-in to processing:

- PVector // 2D/3D vectors
- PShape // shape objects
- ArrayList, IntList, FloatList, StringList // dynamic arrays
- BufferedReader // file IO
- PImage // image data

Imported from libraries:

- Other types (need to import from library to use)
  - **sound → AudioSample, SoundFile ← this lecture**
  - video

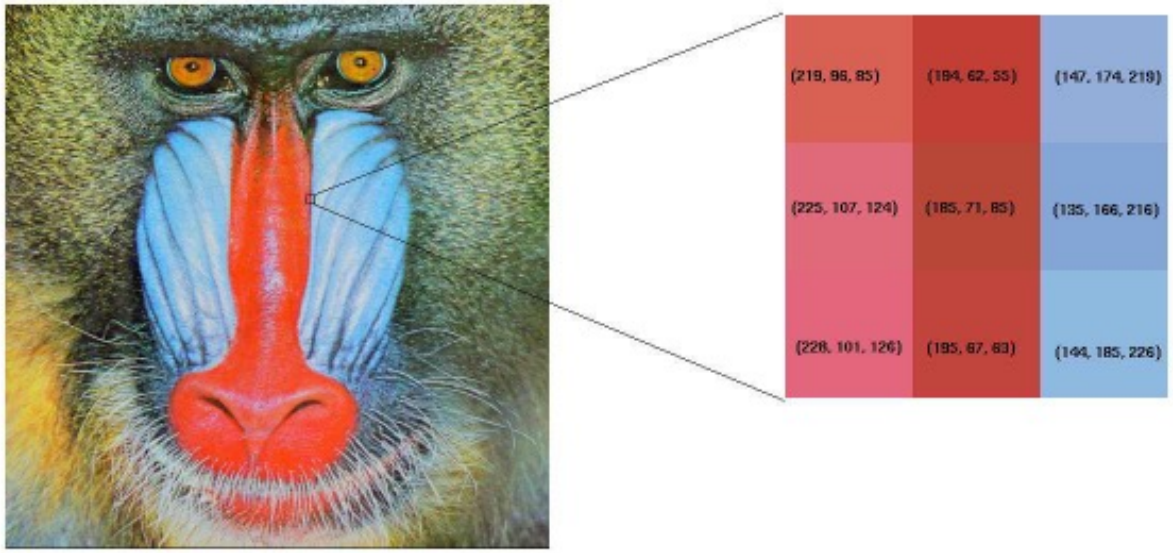
Recall: Audio & Images, as Arrays

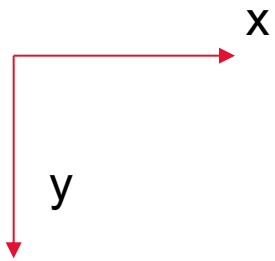
- Audio: 1D (sound samples over time)

1.4	3.5	12	4	0.6	-3.5	-10.3	…

1.4	3.5	12	4	0.6	-3.5	-10.3	…

- Images: 2D (pixel/colour samples over space)





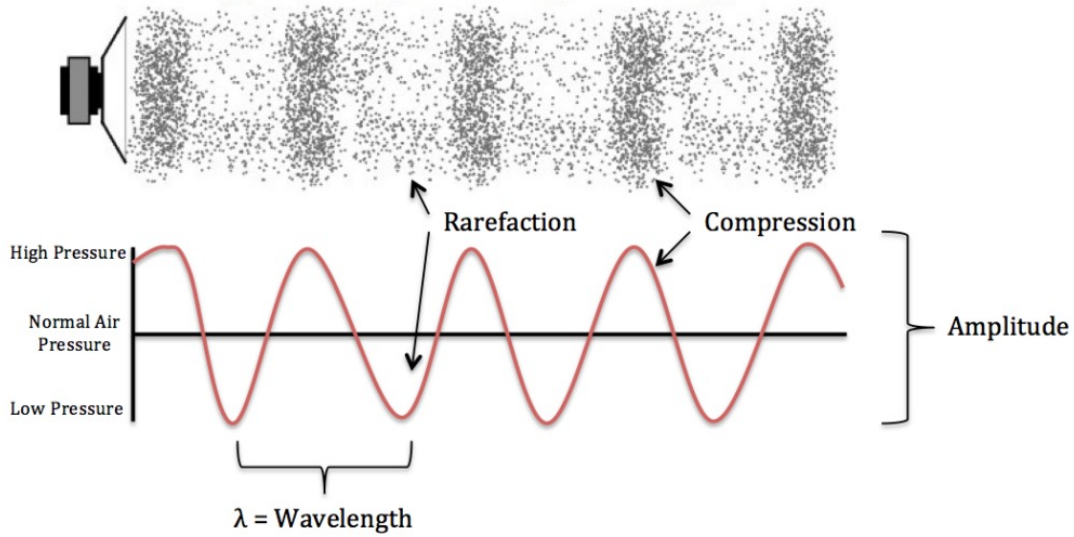
YORK UNIVERSITÉ UNIVERSITY

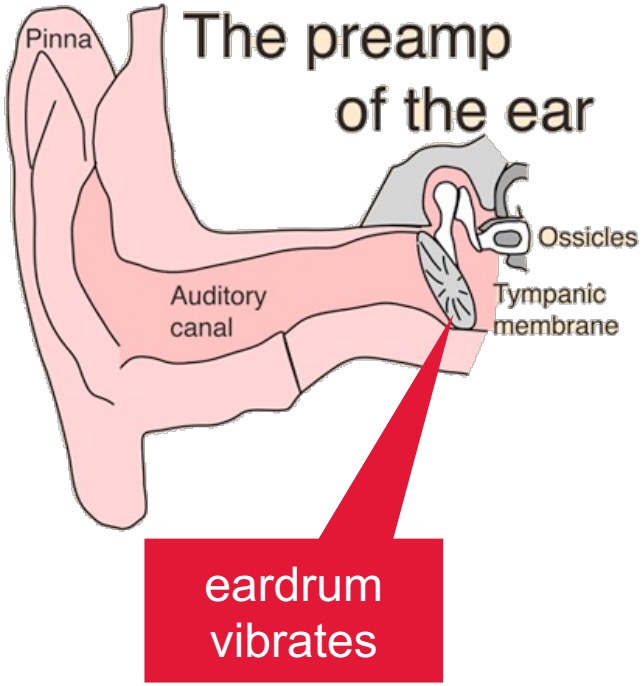
Sound (Audio)

- The perception of the vibration of molecules (pressure waves transmitted through the air)
- Our eardrums oscillate (vibrate) in response to these pressure changes

- Oscillation ←→ Sinusoidal Waves
  - E.g. Concert A (musical note A above middle C) is essentially a sinusoidal wave that oscillates at 440 times per second

Human perception of sound

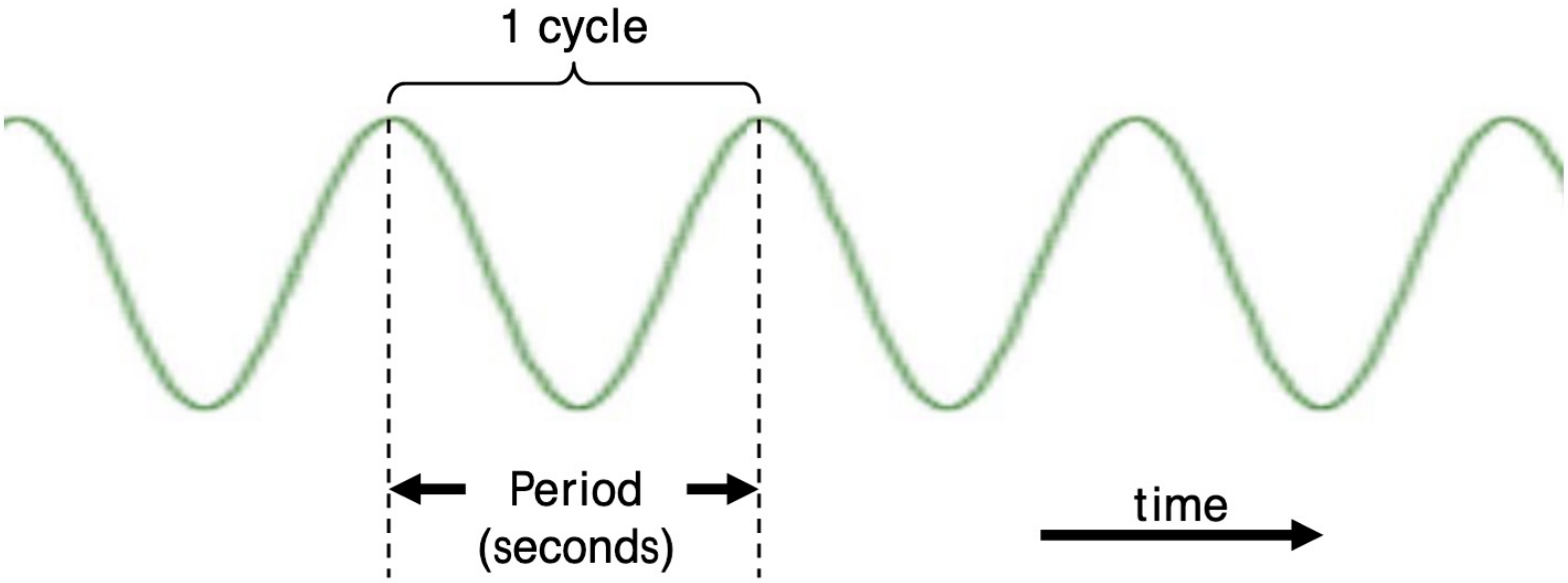




- Occurs through the vibration of molecules in air
- Our eardrums oscillate (vibrate) in response to changes in air pressure

YORK UNIVERSITÉ UNIVERSITY

Sinusoidal Wave (electricity drives speaker)



$$\text{Frequency} = \frac{1}{\text{Period}} \text{ cycles per second (aka Hertz or Hz)}$$

e.g. Frequency = 1/Period => Period = 1/Frequency
Period_{20Hz} = 1 / (20 cycles/sec) = 0.05 sec/cycle
Period_{20kHz} = 1/20000 = 5×10^{-5} sec/cycle

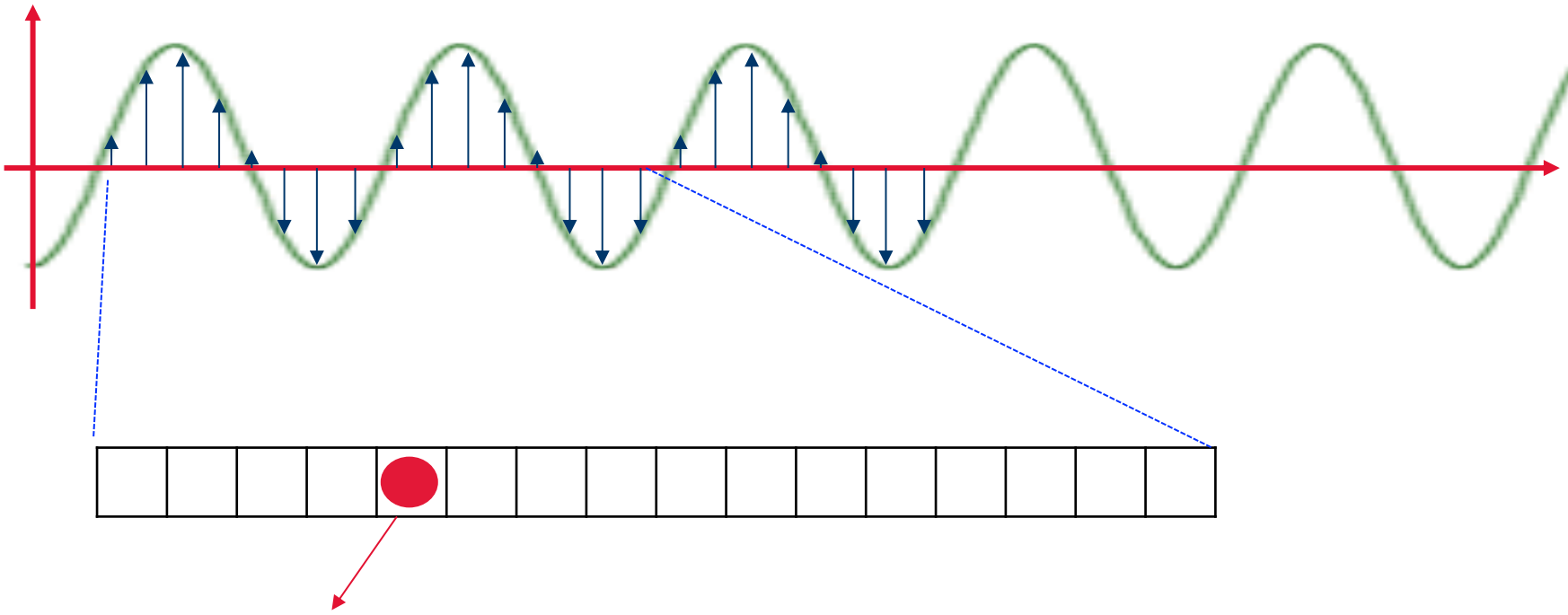
Human perception of audio

- Range: 20 Hz – 20,000 Hz

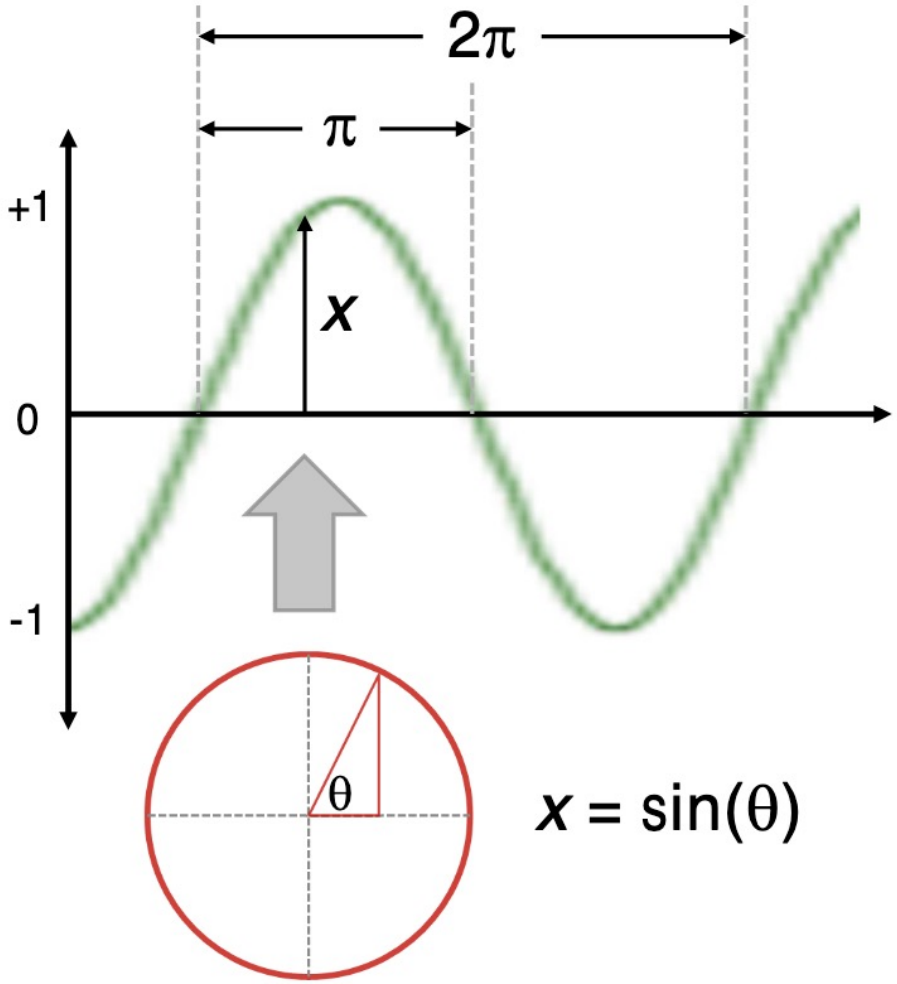
- Speech: up to ≈ 4 kHz
- Music: up to ≈ 20 kHz
- Piano range: 27.5 Hz - 4186 Hz
- Physical property vs. human sensation:

Physical Property	Human Sensation
Amplitude	Volume
Frequency	Pitch

Sinusoidal Wave



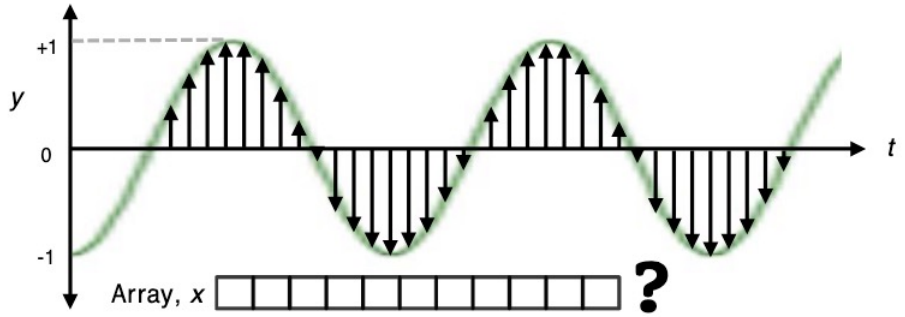
Sine Wave Calculations



Examples

θ	x
0	0.0
π/4	0.707
π/2	1.0
π	0.0
3π/2	-0.707
2π	0.0

Building an Array of Sine Wave Samples



$$y(t) = \sin(2\pi f t), \qquad t = \frac{i}{SR}$$

where y is an array of samples representing a musical note

t is time in seconds

$y(t)$ is the amplitude at time t (between -1 and +1)

f is frequency in cycles per second or Hz

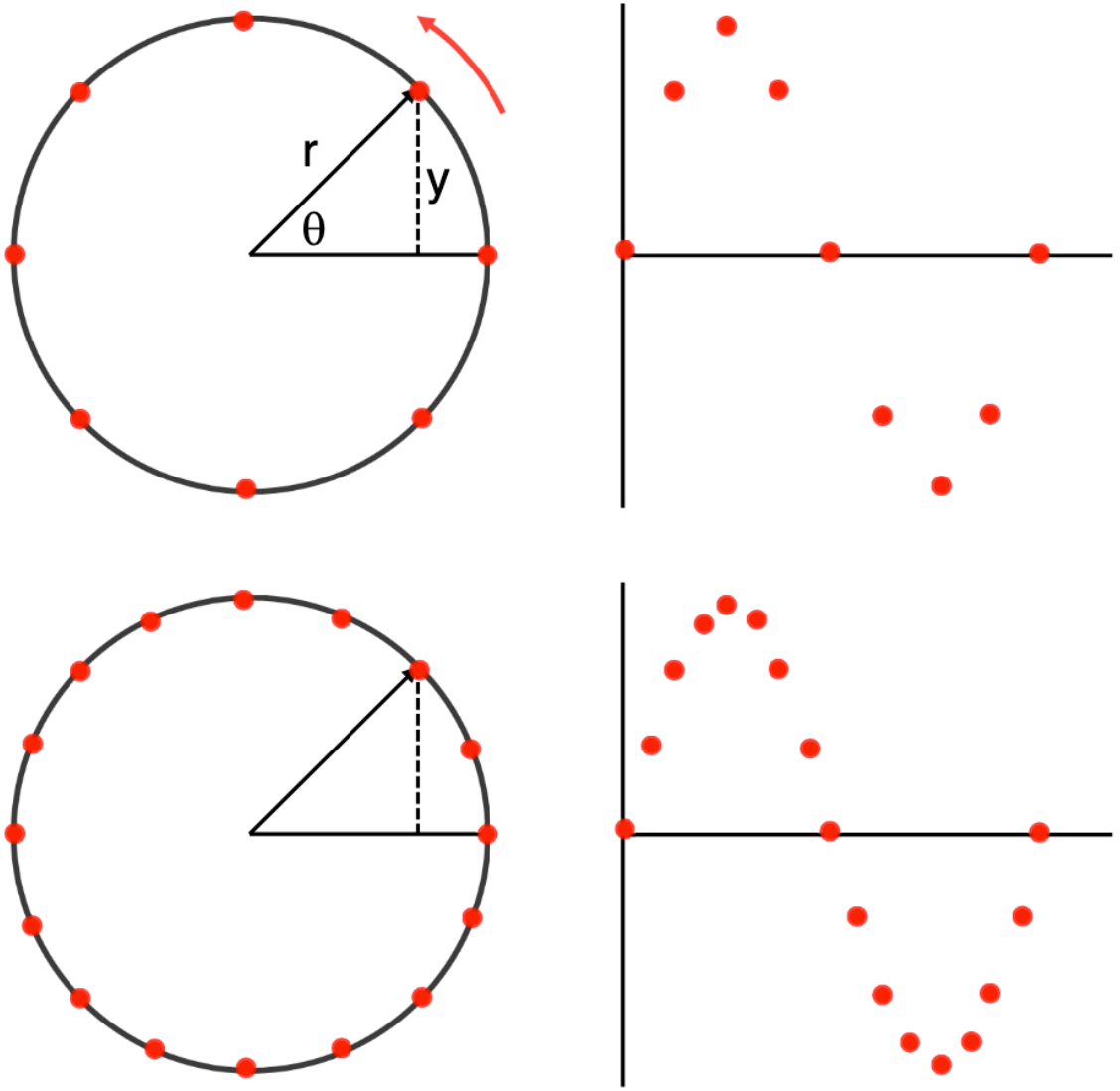
i is an index into the array ($i = 0, 1, 2, \ldots n\text{-}1$)

SR is the sampling rate (number of samples/second)

$$\sin\theta = \frac{y}{r}$$

$$y = r \times \sin\theta$$

Note: $r = 1$



Typical SR (sampling rate) for audio

- SR ~ 44100 Hz (roughly 2x highest perceivable frequency)
  - This ensures there are enough samples to capture the fastest changing tones (sinusoids) in our audio sample
- Note: for speech audio only.. we don't need such a high sampling rate (its highest frequencies only at 4kHz)
  - So SR = 8kHz should be enough
  - The higher the sampling rate, the more samples stored, the more data / memory used!

Example (data usage – uncompressed audio)

1 min of stereo uncompressed audio?

= 1min * 60sec/min * 44100 samples/sec * 2 channels

= 5,292,000 samples (x 4 bytes if float i.e. 32 bits per sample)

= 21,168,000 bytes = 21.168 MB (megabytes) !!

OK, but we have *.mp3 or *.aiff files (compressed)

= a lot smaller (how? throw away parts of the audio that can't be perceived)

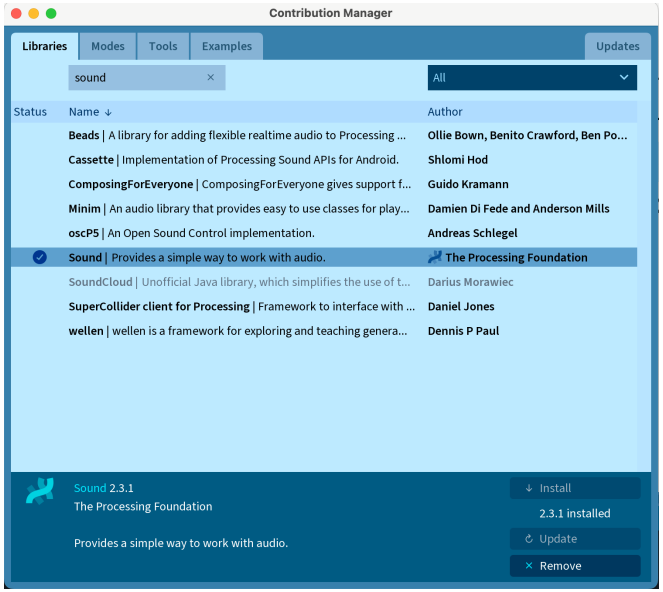
need to be decoded though when read in from a file.. (needs to happen quick)

https://www.colincrawley.com/audio-file-size-calculator/

Synthesizing sound in Processing?

- AudioSample object
  - need to install & import library →

menus→ sketch → Import Library → Manage Libraries



Class Name AudioSample

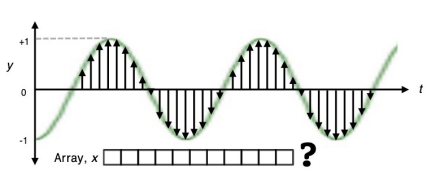
Description This class allows you low-level access to an audio buffer to create, access, manipulate and play back sound samples. If you want to pre-load your audio sample with an audio file from disk you can do so using the SoundFile subclass.

Constructors

```
AudioSample(parent, frames)
AudioSample(parent, frames, stereo)
AudioSample(parent, frames, stereo, frameRate)
AudioSample(parent, frames, frameRate)
AudioSample(parent, data)
AudioSample(parent, data, stereo)
AudioSample(parent, data, frameRate)
AudioSample(parent, data, stereo, frameRate)
```

Parameters

parent	typically use "this"
frames	the desired number of frames for this audiosample
stereo	whether to treat the audiosample as 2-channel (stereo) or not (default: false)
frameRate	the underlying frame rate of the sample (default: 44100)
data	an array of float values to be used as this audiosample's sound data. The audiosample will consequently have as many frames as the length of the given array.



$$y(t) = \sin(2\pi f t), \qquad t = \frac{i}{SR}$$

where y is an array of samples representing a musical note
t is time in seconds
$y(t)$ is the amplitude at time t (between -1 and +1)
f is frequency in cycles per second or Hz
i is an index into the array ($i = 0, 1, 2, \ldots n\text{-}1$)
SR is the sampling rate (number of samples/second)

```
import processing.sound.*;

// CREATING A SIMPLE TONE
// (pure sine/cosine waveform)

AudioSample sample;

void setup() {
  size(640, 360);
  background(255);

  // Create an array of sinusoid y(t)
  int sampleRate = 44100;    // number of samples per cycle = SR
  float freq = 440;          // frequency in Hz (cycles/sec)

  float[] sinewave = new float[sampleRate];

  for (int i = 0; i < sampleRate; i++) {
    sinewave[i] = sin(TWO_PI*freq*i/sampleRate);  // formula for y(t) above right
  }

  // Create audiosample from data, set framerate
  sample = new AudioSample(this, sinewave, sampleRate);

  // Play the sample in a loop (but don't make it too loud)
  sample.amp(0.5);      // sets to half amplitude
  sample.loop();        // audio buffer setup to play over and over in loop
}

void draw() {
}
```

YORK UNIVERSITÉ UNIVERSITY

AudioSample (supported methods/behaviors)

Methods

`amp()`	Changes the amplitude/volume of the player.
`channels()`	Returns the number of channels in the audiosample as an int (1 for mono, 2 for stereo).
`cue()`	Cues the playhead to a fixed position in the audiosample.
`cueFrame()`	Cues the playhead to a fixed position in the audiosample.
`duration()`	Returns the duration of the audiosample in seconds.
`frames()`	Returns the number of frames of the audiosample as an int.
`jump()`	Jump to a specific position in the audiosample while continuing to play (or starting to play if it wasn't playing already).
`jumpFrame()`	Jump to a specific position in the audiosample without interrupting playback.
`loop()`	Starts the playback of the audiosample.
`play()`	Starts the playback of the audiosample.
`playFor()`	Starts the playback of the audiosample for the specified duration or to the end of the audiosample, whichever comes first.
`rate()`	Set the relative playback rate of the audiosample.
`resize()`	Resizes the underlying buffer of the audiosample to the given number of frames.
`sampleRate()`	Returns the underlying sample rate of the audiosample.
`pan()`	Pan the soundfile in a stereo panorama.
`set()`	Set multiple parameters at once.
`stop()`	Stops the playback.
`position()`	Get current sound file playback position in seconds.
`positionFrame()`	Get frame index of current sound file playback position.
`percent()`	Get current sound file playback position in percent.
`pause()`	Stop the playback of the sample, but cue it to the current position.
`read()`	The underlying data of the audiosample can be read and written in several different.
`write()`	The underlying data of the audiosample can be read and (over)written in several different ways.

```
// Create the audiosample based on the data in sinewave[] array
// set framerate to play at 44100Hz

  sample = new AudioSample(this, sinewave, sampleRate);

// output some info

  println("\t duration = " + sample.duration() + " secs");
  println("\t SR       = " + sample.sampleRate() + " samples/sec");
  println("\t channels = " + sample.channels());
  println("\t frames   = " + sample.frames());



// Play the sample in a loop (but don't make it too loud)

  sample.amp(0.2);
  sample.loop();
```

SoundFile (load & playback an existing sound file)

Class Name SoundFile

Description This is a Soundfile player which allows to play back and manipulate sound files. Supported formats are: WAV, AIF/AIFF, and MP3. MP3 decoding can be very slow on ARM processors (Android/Raspberry Pi), we generally recommend you use lossless WAV or AIF files.

Constructors

```
SoundFile(parent, path)
SoundFile(parent, path, cache)
```

Parameters

parent	typically use "this"
path	filename of the sound file to be loaded
cache	keep the sound data in RAM once it has been decoded (default: true). Note that caching essentially disables garbage collection for the SoundFile data, so if you are planning to load a large number of audio files, you should set this to false.

Methods

`removeFromCache()`	Remove this SoundFile's decoded audio sample from the cache, allowing it to be garbage collected once there are no more references to this SoundFile.
`channels()`	Returns the number of channels of the soundfile as an int (1 for mono, 2 for stereo).
`cue()`	Cues the playhead to a fixed position in the soundfile.
`duration()`	Returns the duration of the soundfile in seconds.
`frames()`	Returns the number of frames of this soundfile.
`play()`	Starts the playback of the soundfile.
`jump()`	Jump to a specific position in the soundfile while continuing to play (or starting to play if it wasn't playing already).
`pause()`	Stop the playback of the file, but cue it to the current position.
`isPlaying()`	Check whether this soundfile is currently playing.
`loop()`	Starts playback which will loop at the end of the soundfile.
`amp()`	Changes the amplitude/volume of the player.
`pan()`	Move the sound in a stereo panorama.
`rate()`	Set the playback rate of the soundfile.
`stop()`	Stops the playback.

YORK UNIVERSITÉ UNIVERSITY

```
import processing.sound.*;

// Playing/loading an AudioSample from a SoundFile
SoundFile file;

void setup() {
  size(640, 360);
  background(255);

  // Load a soundfile from the /data folder of the sketch and play it back
  file = new SoundFile(this, "fantasy.wav");                // sample 1 (music, stereo)
  //file = new SoundFile(this, "preamble.wav");             // sample 2 (speech, mono)

}

void draw() {
}

// use mouse clicks to toggle play/pause
void mousePressed() {
  if (file.isPlaying() ) {
      file.pause();
      println("..paused");
  }
  else {
   file.play();
   println("playing..");
  }
}
```

Note

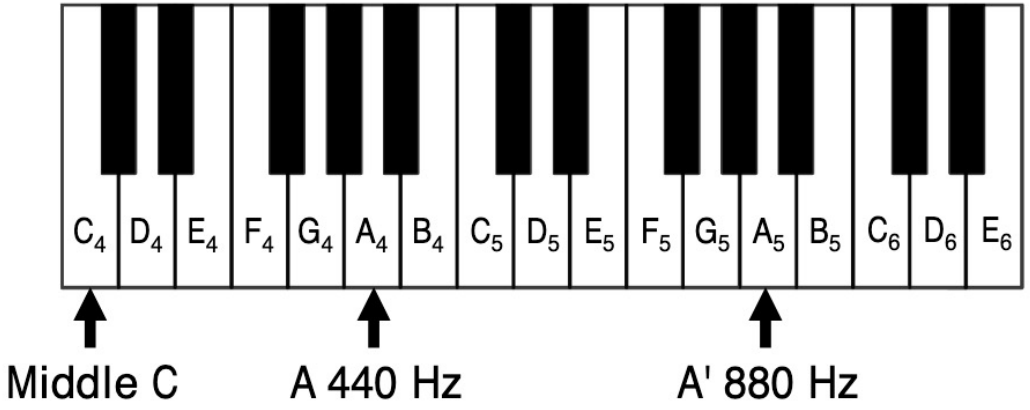
- Any files used by the program, need to reside inside the same folder as the sketch being run!
  - In this example, “fantasy.wav” and “preamble.wav” are located in the folder with the same name as the sketch
  - This goes for *any* resources used by your program

Equal-Tempered Music Scale

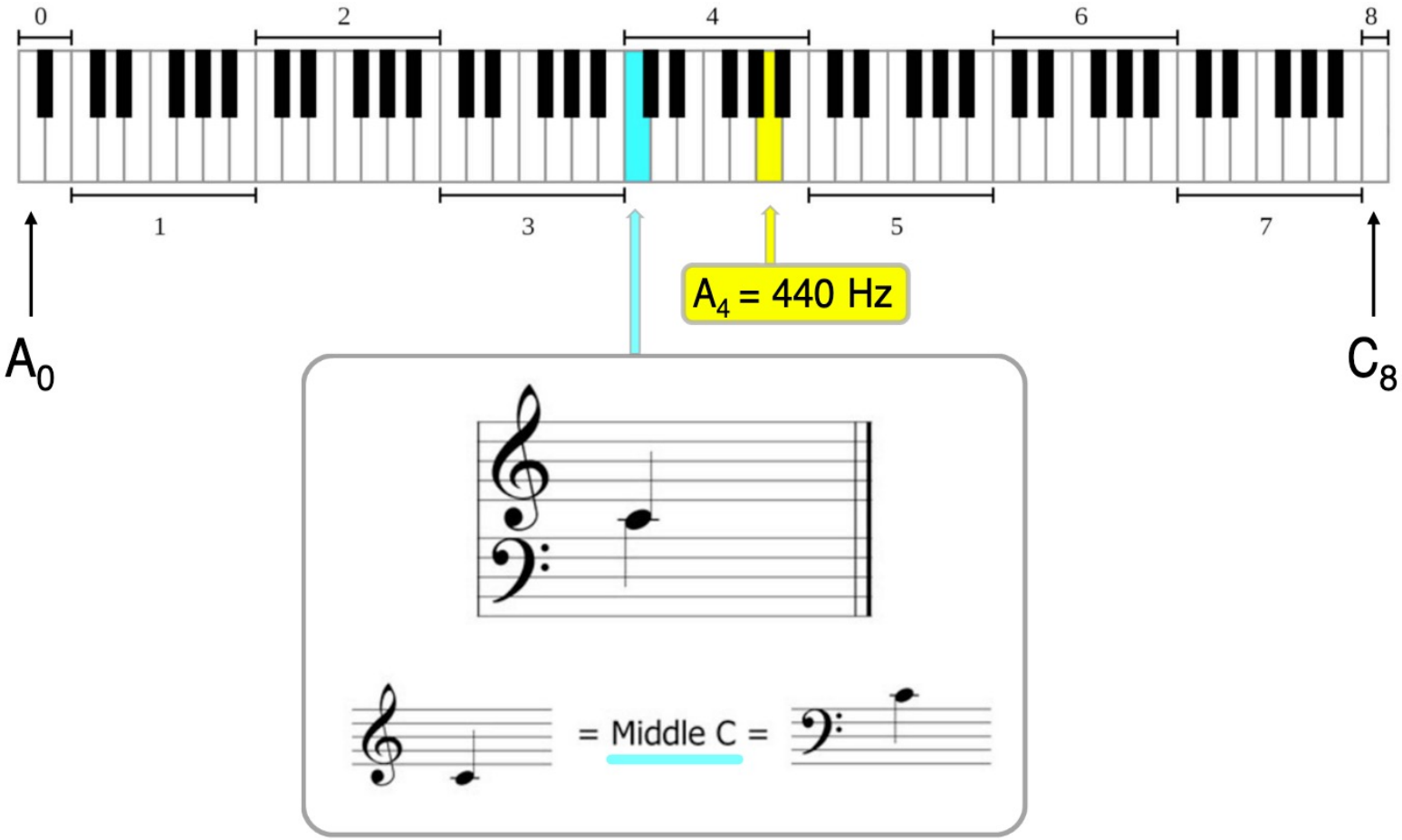
The harmonic system used in most Western music is based on the "equal-tempered scale", as typified by the pattern of white and black keys on a piano. The frequency of each note on a piano keyboard is related to its neighbor by the following formula:

$$f_{n+1} = f_n \times 2^{1/12}$$

where note $n + 1$ is "one semitone" above note n. The factor $2^{1/12}$ ensures that two notes separated by twelve steps in the equal-tempered scale differ in frequency by a factor of $2^{12/12} = 2$. This interval is known as an "octave". To allow musicians to travel and perform with different orchestras in different countries, a standard evolved and was adopted in the early 1900s to ensure instruments were in tune with each other. The standard specifies that "A above middle C" has a frequency of 440 Hz (see below). With this reference point, and with the $2^{1/12}$ frequency factor between adjacent notes, the frequency of any note on any instrument was standardized.

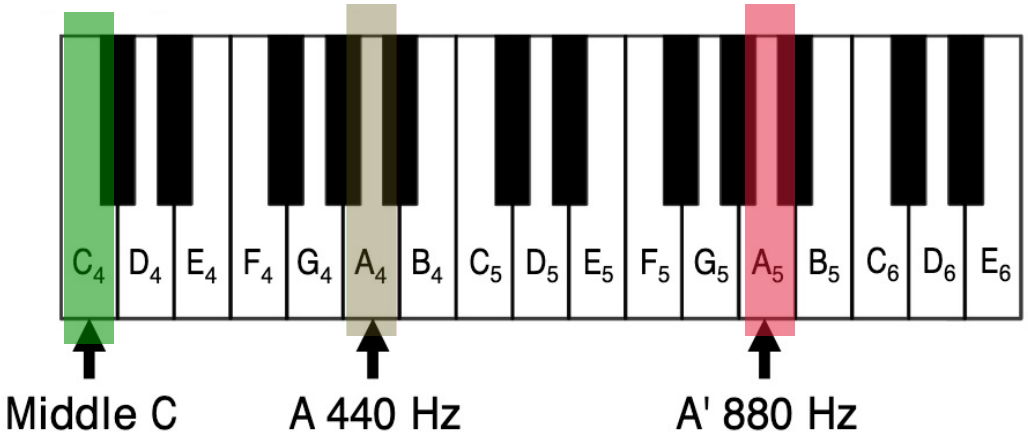


Musical Notes on a Piano



Lab 5

- Involves generating specific tones relating to notes on a keyboard/piano
- Beginning with a reference tone (say 440Hz)
  - if you know the note you want (octave relative to reference octave, and position of note (relative to reference note)
  - you can calculate the frequency of the note you want to generate
  - you can use that frequency value as your variable to generate the right sinusoid… which when played back should sound like the correct tone



$$f_{n+1} = f_n \times 2^{1/12}$$

$f_{desired} = f_{ref} * 2^{s/12}$

$f_{desired} = f_{A5} = f_{A4} * 2^{+12/12}$

$= 440 * 2$

$= 880Hz$

← A5 is +12 semi-tones from A4 (i.e. up one octave)

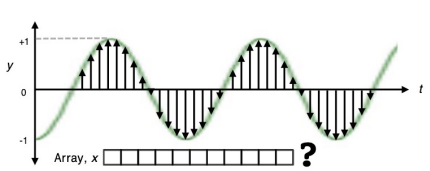
$f_{desired} = f_{C4} = f_{A4} * 2^{-9/12}$

$= 440 * 0.5946$

$= 261.63 Hz$

← C4 is -9 semi-tones from A4 (i.e. down 9/12 octave)

YORK UNIVERSITÉ UNIVERSITY



$$y(t) = \sin(2\pi f t), \qquad t = \frac{i}{SR}$$

where y is an array of samples representing a musical note
t is time in seconds
$y(t)$ is the amplitude at time t (between -1 and +1)
f is frequency in cycles per second or Hz
i is an index into the array ($i = 0, 1, 2, \ldots n\text{-}1$)
SR is the sampling rate (number of samples/second)

```
import processing.sound.*;

// CREATING A SIMPLE TONE
// (pure sine/cosine waveform)

AudioSample sample;

void setup() {
  size(640, 360);
  background(255);

  // Create an array of sinusoid y(t)
  int sampleRate = 44100;    // number of samples per cycle = SR
  float freq = 440;          // replace with freq relative to ref freq!

  float[] sinewave = new float[sampleRate];

  for (int i = 0; i < sampleRate; i++) {
    sinewave[i] = sin(TWO_PI*freq*i/sampleRate);  // formula for y(t) above right
  }

  // Create audiosample from data, set framerate
  sample = new AudioSample(this, sinewave, sampleRate);

  // Play the sample in a loop (but don't make it too loud)
  sample.amp(0.5);      // sets to half amplitude
  sample.loop();        // audio buffer setup to play over and over in loop
}

void draw() {
}
```

YORK UNIVERSITÉ UNIVERSITY