Sound Processing with Py5

A Comprehensive Guide

Course Topics:

- Audio Fundamentals
- Sound Playback & Control
- Sound Analysis & Visualization
- Sound Synthesis
- Audio Effects
- Real-time Processing

What You'll Learn

✅ Load and play audio files
✅ Control sound (volume, rate, pan)
✅ Analyze audio (amplitude, frequency)
✅ Synthesize sounds (oscillators)
✅ Apply effects (filters, reverb, delay)
✅ Process microphone input
✅ Create interactive visualizations

Prerequisites: Basic Python and Py5 knowledge

Course Outline

1. **Setup & Installation**
2. **Sound Fundamentals**
3. **Basic Playback**
4. **Sound Control**
5. **Sound Analysis**
6. **Sound Synthesis**
7. **Audio Effects**
8. **Microphone Input**
9. **Complete Projects**

Part 1: Setup & Installation

Installing Py5

```
# Install py5
pip install py5

# Verify installation
python -c "import py5; print(py5.__version__)"
```

System Requirements:

- Python 3.8+
- Working audio output
- Microphone (optional, for input)

Installing the Sound Library

```
import py5_tools

# Download the Processing Sound library
py5_tools.processing.download_library("Sound")

# Check installed libraries
print(py5_tools.processing.installed_libraries())
# Output: ['Sound', ...]
```

This only needs to be done once!

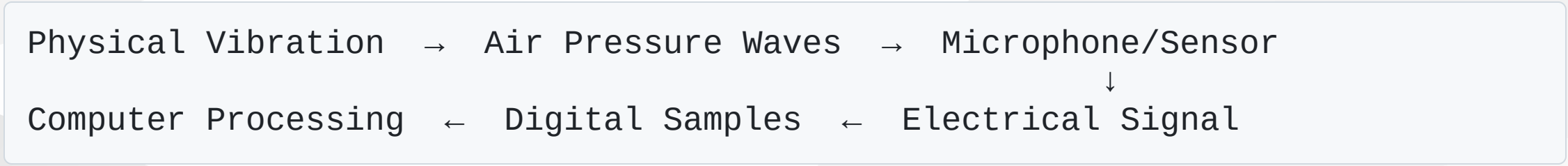
Project Structure

```
my_sound_project/
├── sketch.py           # Your main code
└── data/               # Audio files folder
    ├── music.mp3
    ├── sound.wav
    └── beat.aiff
```

Supported formats: WAV, MP3, AIFF

Part 2: Sound Fundamentals

What is Sound?



Key Concepts:

- **Amplitude**: Loudness (volume)
- **Frequency**: Pitch (Hz)
- **Sample Rate**: Quality (44100 Hz = CD quality)

Digital Audio Representation

```
Continuous Sound Wave
         ↓
    Sampling (44,100 times per second)
         ↓
 Discrete Samples (numbers)
         ↓
   Quantization (16-bit values)
         ↓
  Digital Audio File (.wav, .mp3)
         ↓
 Digital-to-Analog Conversion
         ↓
   Speakers → Sound!
```

Sample Rate: How many samples per second (Hz)

Bit Depth: How many values per sample (16-bit = 65,536 levels)

Sound Properties Diagram

Amplitude

Volume/Loudness

Frequency

Pitch (Hz)

Duration

Length in Time

Phase

Wave Timing

Part 3: Basic Playback

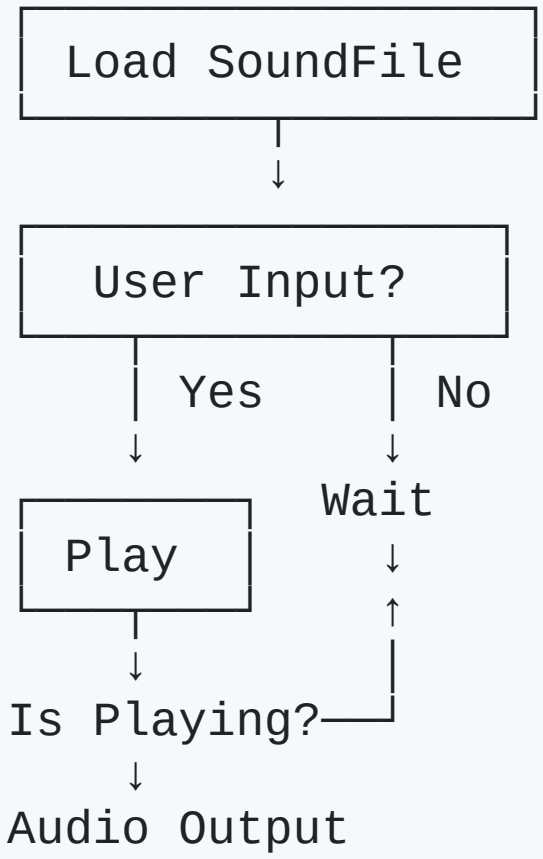
Your First Sound Sketch

```
import py5
from processing.sound import SoundFile

soundfile = None

def setup():
    global soundfile
    py5.size(400, 300)

    # Load sound from data folder
    soundfile = SoundFile(py5.get_current_sketch(),
                          "mysound.wav")
    soundfile.play()

def draw():
    py5.background(220)
    py5.text("Playing sound!", 50, 150)

py5.run_sketch()
```

Basic Playback Methods

Method	Description
`.play()`	Play once
`.loop()`	Play repeatedly
`.stop()`	Stop playback
`.pause()`	Pause (resume with play)
`.jump(seconds)`	Jump to position
`.isPlaying()`	Check if playing

Sound Playback Flow



Part 4: Sound Control

Volume Control (Amplitude)

```
def draw():
    # Map mouse X to volume (0.0 to 1.0)
    volume = py5.remap(py5.mouse_x, 0, py5.width, 0.0, 1.0)
    soundfile.amp(volume)

    py5.background(220)
    py5.text(f"Volume: {volume:.2f}", 50, 100)

    # Visual feedback
    py5.rect(50, 150, volume * 300, 30)
```

Range: 0.0 (silent) to 1.0 (full volume)

Playback Rate Control

```
def draw():
    # Map mouse Y to playback rate
    rate = py5.remap(py5.mouse_y, 0, py5.height, 0.5, 2.0)
    soundfile.rate(rate)

    py5.text(f"Rate: {rate:.2f}x", 50, 100)
```

Effects:

- `rate < 1.0` : Slower, lower pitch
- `rate = 1.0` : Normal speed
- `rate > 1.0` : Faster, higher pitch

Stereo Panning

```
def draw():
    # Pan from left (-1.0) to right (1.0)
    pan = py5.remap(py5.mouse_x, 0, py5.width, -1.0, 1.0)
    soundfile.pan(pan)

    py5.text(f"Pan: {pan:.2f}", 50, 100)
    py5.text("Left ← → Right", 50, 130)
```

-1.0

Left Speaker

🔊

0.0

Center

🔊🔊

+1.0

Right Speaker

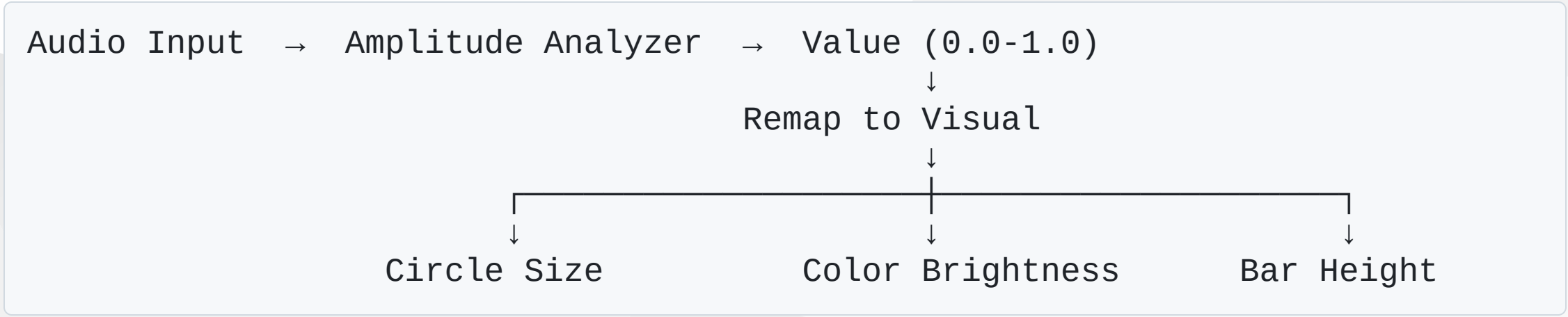
Part 5: Sound Analysis

Amplitude Analysis

What it measures: Overall volume/loudness of audio

```
from processing.sound import Amplitude

amplitude = Amplitude(py5.get_current_sketch())
amplitude.input(soundfile)

def draw():
    amp = amplitude.analyze()  # Returns 0.0 to 1.0
    circle_size = py5.remap(amp, 0, 0.5, 50, 400)
    py5.circle(width/2, height/2, circle_size)
```

Amplitude Visualization Flow



FFT - Frequency Analysis

FFT = Fast Fourier Transform

Breaks audio into frequency bands (low, mid, high)

Error 400: Unable to decode the source. The source is not in valid Base64 scheme.

FFT Code Example

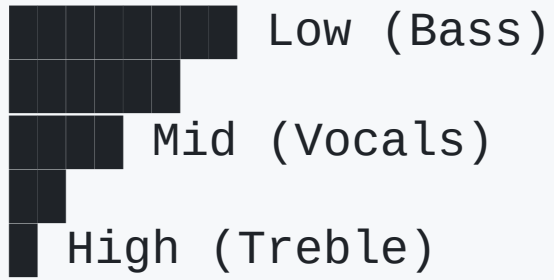
```
from processing.sound import FFT

fft = FFT(py5.get_current_sketch(), 512)  # 512 bands
fft.input(soundfile)

def draw():
    fft.analyze()

    for i in range(512):
        band_amplitude = fft.spectrum[i]
        bar_height = py5.remap(band_amplitude, 0, 0.5,
                               0, py5.height)
        py5.rect(i * 2, py5.height - bar_height, 2, bar_height)
```

FFT Spectrum Visualization

Frequency Bands:

- 20-60 Hz: Sub-bass
- 60-250 Hz: Bass
- 250-2000 Hz: Midrange
- 2000-6000 Hz: Presence
- 6000-20000 Hz: Brilliance

Visual Bars:



Left side = Low frequencies

Right side = High frequencies

Waveform Visualization

Shows the actual shape of the sound wave

```
from processing.sound import Waveform

waveform = Waveform(py5.get_current_sketch(), 512)
waveform.input(soundfile)

def draw():
    waveform.analyze()

    for i in range(512):
        x = py5.remap(i, 0, 512, 0, py5.width)
        y = py5.remap(waveform.data[i], -1, 1,
                      0, py5.height)
        py5.point(x, y)
```

Part 6: Sound Synthesis

What is Sound Synthesis?

Creating sound from scratch using code!

```
Algorithm  →  Oscillator  →  Generate Waveform  →  Audio Output
```

Key Component: Oscillator - generates repeating waveforms

Oscillator Types

Sine Wave

Pure, smooth tone

Triangle Wave

Softer, mellow

Sawtooth Wave

Bright, buzzy

Square Wave

Harsh, retro game sound

Each has a unique timbre (tone quality)!

Waveform Shapes

Sine Wave: Smooth, pure tone

Square Wave: Electronic, retro game sound

Creating a Sine Wave

```
from processing.sound import SinOsc

sine = None

def setup():
    global sine
    py5.size(600, 400)
    sine = SinOsc(py5.get_current_sketch())
    sine.freq(440)  # A4 note (440 Hz)
    sine.amp(0.5)   # 50% volume

def mouse_pressed():
    sine.play()  # Start oscillator

def mouse_released():
    sine.stop()   # Stop oscillator
```

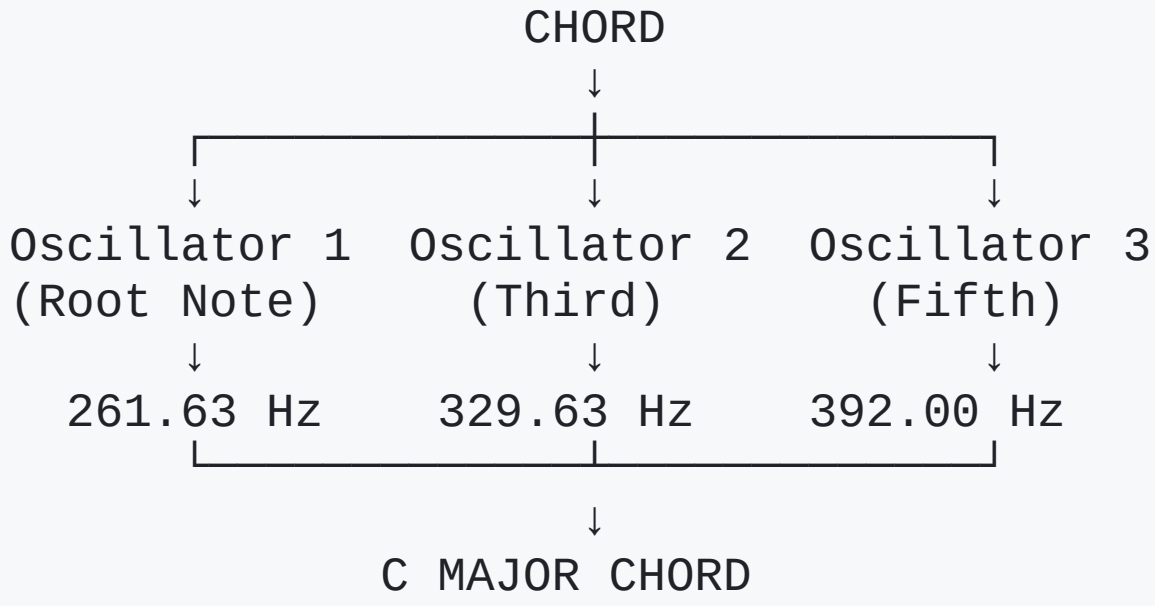
Musical Note Frequencies

Note	Frequency (Hz)	Use Case
C4	261.63	Middle C
A4	440.00	Tuning reference
C5	523.25	Higher octave

Formula for octaves:

- Up one octave: frequency × 2
- Down one octave: frequency ÷ 2

Building Chords



C Major = C (261.63) + E (329.63) + G (392.00)

Chord Example Code

```
from processing.sound import SinOsc

osc1 = SinOsc(py5.get_current_sketch())
osc2 = SinOsc(py5.get_current_sketch())
osc3 = SinOsc(py5.get_current_sketch())

# C Major chord
osc1.freq(261.63)  # C
osc2.freq(329.63)  # E
osc3.freq(392.00)  # G

def key_pressed():
    if py5.key == ' ':
        osc1.play()
        osc2.play()
        osc3.play()
```

Part 7: Audio Effects

Effect Types Overview

FILTERS

- Low-Pass (Remove highs)
- High-Pass (Remove lows)
- Band-Pass (Isolate range)

TIME-BASED

- Delay (Echo)
- Reverb (Space/Room)

Filter Types Diagram

Low-Pass Filter

- ✅ Low frequencies PASS
- ❌ High frequencies BLOCKED
- **Effect:** Muffled, underwater sound
- **Example:** Phone voice quality

High-Pass Filter

- ❌ Low frequencies BLOCKED
- ✅ High frequencies PASS
- **Effect:** Thin, tinny sound
- **Example:** Old radio

Low-Pass Filter

Removes high frequencies (treble)

Creates muffled, underwater effect

```
from processing.sound import LowPass

lowpass = LowPass(py5.get_current_sketch())
lowpass.process(soundfile)

def draw():
    # Cutoff frequency: 100 Hz to 10000 Hz
    cutoff = py5.remap(py5.mouse_x, 0, py5.width,
                       100, 10000)
    lowpass.freq(cutoff)

    py5.text(f"Cutoff: {cutoff:.0f} Hz", 50, 100)
```

High-Pass Filter

Removes low frequencies (bass)

Creates thin, tinny effect

```
from processing.sound import HighPass

highpass = HighPass(py5.get_current_sketch())
highpass.process(soundfile)

def draw():
    cutoff = py5.remap(py5.mouse_x, 0, py5.width,
                       100, 5000)
    highpass.freq(cutoff)
```

Reverb Effect

Simulates room acoustics and space

Controls:

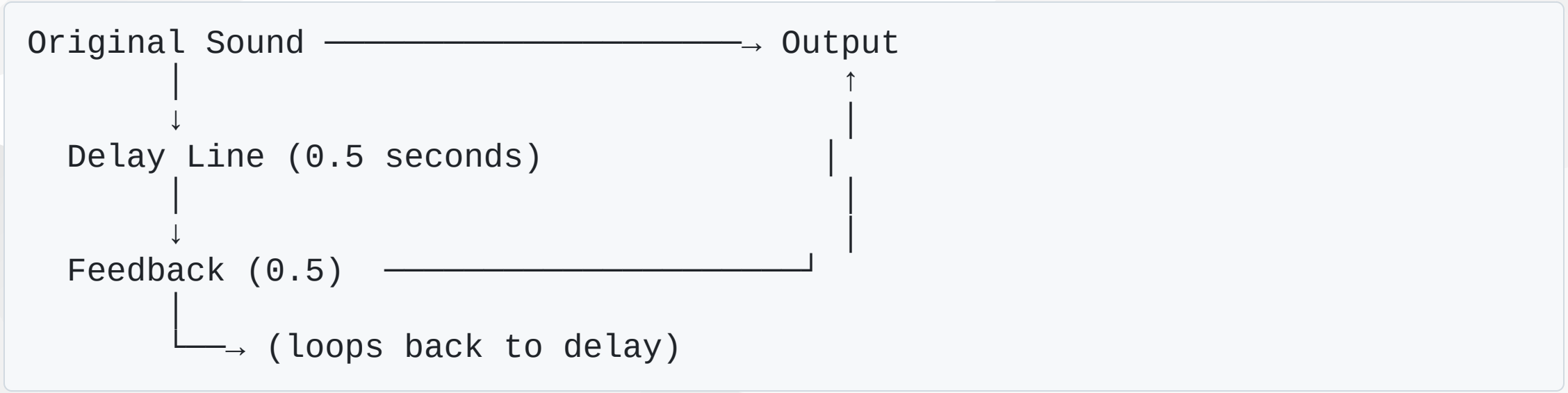
- Room size (0.0 = small room, 1.0 = huge hall)
- Damping (0.0 = bright, 1.0 = dark/muffled)
- Wet/Dry mix

Reverb Code

```
from processing.sound import Reverb

reverb = Reverb(py5.get_current_sketch())
reverb.process(soundfile)

def draw():
    # Room size: 0.0 (small) to 1.0 (huge)
    room = py5.remap(py5.mouse_x, 0, py5.width, 0.0, 1.0)
    reverb.room(room)

    # Damping: 0.0 (bright) to 1.0 (dark)
    damp = py5.remap(py5.mouse_y, 0, py5.height, 0.0, 1.0)
    reverb.damp(damp)
```

Delay/Echo Effect

Repeats sound with time offset



Controls:

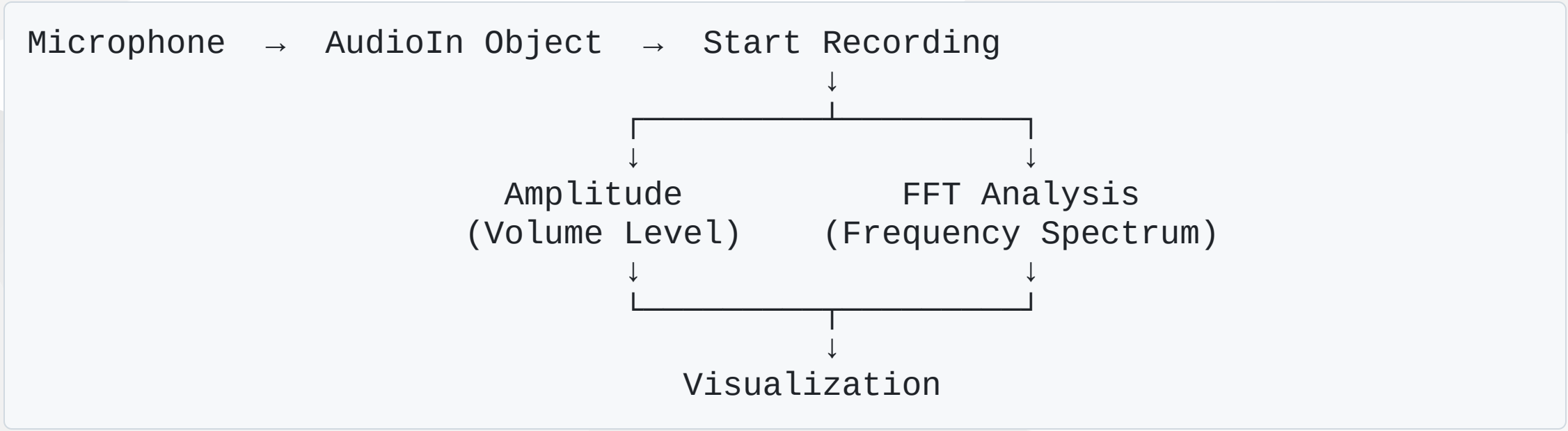
- Delay time (seconds between repeats)
- Feedback amount (how many repeats)

Delay Code

```
from processing.sound import Delay

delay = Delay(py5.get_current_sketch())
delay.process(soundfile)

def draw():
    # Delay time in seconds
    time = py5.remap(py5.mouse_x, 0, py5.width, 0.0, 2.0)
    delay.time(time)

    # Feedback: how much repeats
    feedback = py5.remap(py5.mouse_y, 0, py5.height,
                         0.0, 0.9)
    delay.feedback(feedback)
```

Part 8: Microphone Input

Audio Input Flow



Basic Microphone Input

```
from processing.sound import AudioIn, Amplitude

mic = None
amplitude = None

def setup():
    global mic, amplitude
    py5.size(600, 400)

    # Create microphone input
    mic = AudioIn(py5.get_current_sketch(), 0)
    mic.start()  # Start listening

    # Analyze microphone
    amplitude = Amplitude(py5.get_current_sketch())
    amplitude.input(mic)
```

Microphone Volume Meter

```
def draw():
    py5.background(0)

    # Get microphone level
    level = amplitude.analyze()

    # Visualize as growing circle
    size = py5.remap(level, 0, 0.5, 50, 400)

    py5.fill(100, 255, 150)
    py5.circle(py5.width/2, py5.height/2, size)

    py5.fill(255)
    py5.text(f"Mic Level: {level:.3f}", 20, 30)
    py5.text("Make some noise!", 20, 50)
```

Microphone FFT Spectrum

```
from processing.sound import AudioIn, FFT

mic = AudioIn(py5.get_current_sketch(), 0)
mic.start()

fft = FFT(py5.get_current_sketch(), 256)
fft.input(mic)

def draw():
    py5.background(0)
    fft.analyze()

    for i in range(256):
        height = py5.remap(fft.spectrum[i], 0, 0.5,
                           0, py5.height)
        py5.rect(i * 3, py5.height - height, 3, height)
```

Part 9: Complete Projects

Project 1: Music Visualizer

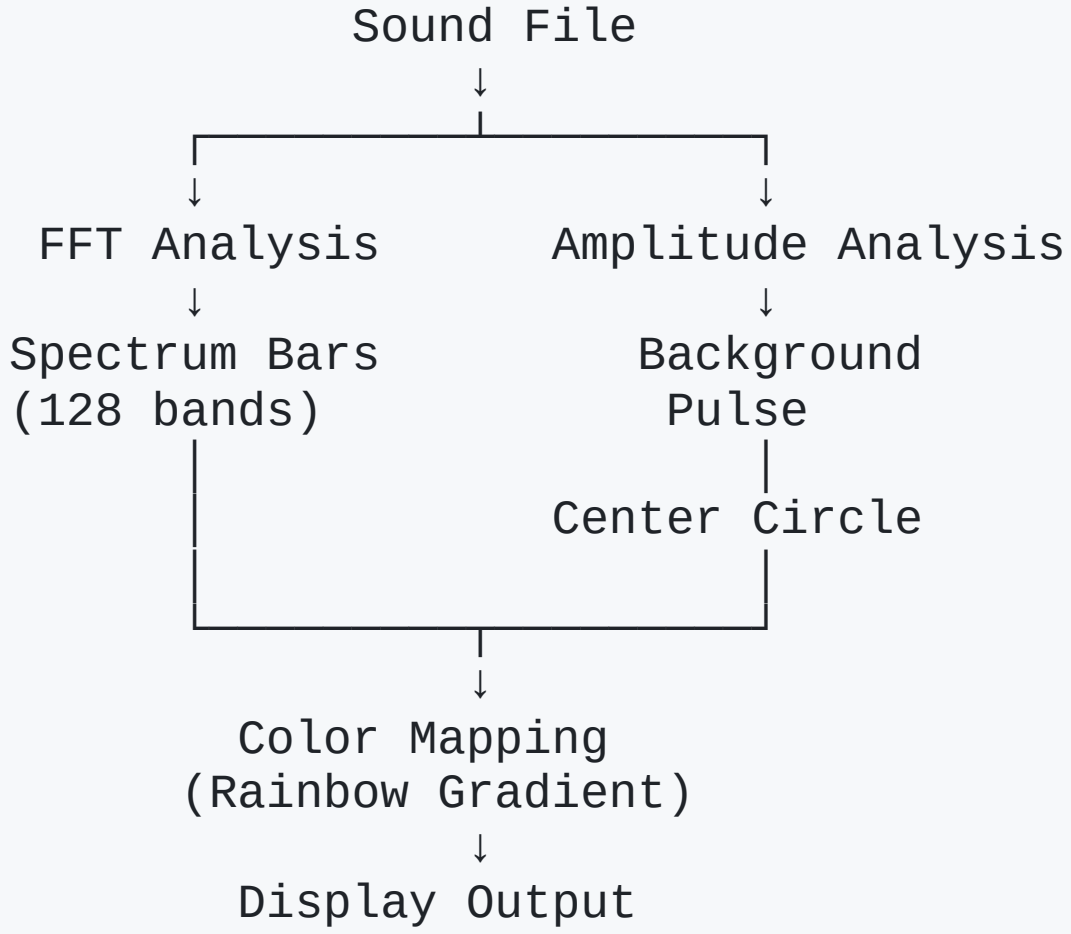
Features:

- FFT spectrum bars
- Amplitude-reactive background
- Rainbow colors
- Smooth animation

Techniques:

- Multiple analysis types
- Color mapping
- Smoothing algorithms

Visualizer Architecture



Project 2: Interactive Synthesizer

Features:

- 8 playable notes
- Visual keyboard
- Multiple oscillator types
- Real-time frequency display

Keys: A S D F G H J K

```
Key Press  →  Note Frequency  →  Oscillator  →  Audio Output
    ↓
Visual Highlight
```

Project 3: Microphone Visualizer

Features:

- Circular spectrum display
- 360° frequency visualization
- Pulsing center
- Real-time responsiveness

Technique: Polar coordinates

```
x = cos(angle) × radius
y = sin(angle) × radius
```

Part 10: Best Practices

Performance Tips

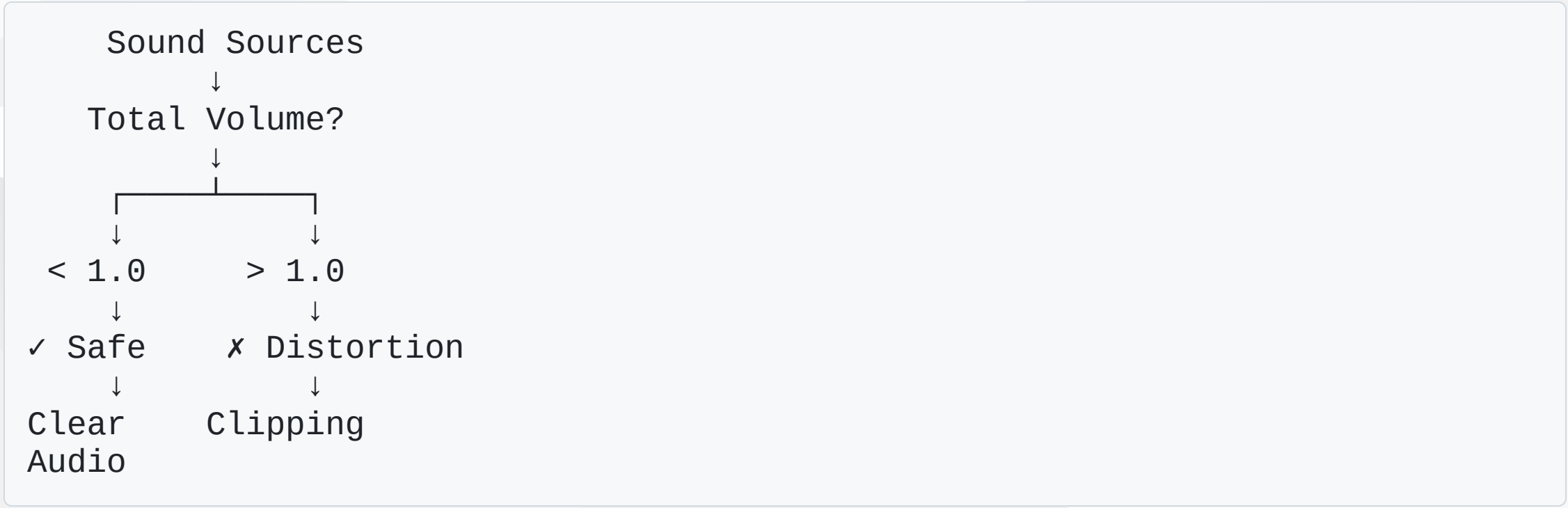
✅ **DO:**

- Use 128-256 FFT bands (not 512+)
- Stop sounds when not needed
- Limit simultaneous sounds to 3-5
- Use appropriate frame rates (30-60 FPS)

❌ **DON'T:**

- Load files in `draw()` (only in `setup()`)
- Create new analyzers every frame
- Forget to call `.stop()` on sounds

Volume Management



Best Practice: Start with 0.3-0.5 amplitude

Common Issues & Solutions

Problem	Solution
No sound	Check data folder path
Distortion	Lower amplitude
Lag	Reduce FFT bands
Crackling	Check sample rate
No mic input	Verify permissions

File Format Recommendations

Format	Pros	Cons	Use Case
WAV	High quality, fast	Large files	Development
MP3	Small files	Compression	Distribution
AIFF	High quality	Mac-friendly	Professional

Recommendation: Use WAV during development, convert to MP3 for sharing

Project Structure Best Practices

```
sound_project/
├── main.py                 # Main sketch
├── config.py               # Settings/constants
├── visualizers.py          # Visual components
├── README.md               # Documentation
└── data/
    ├── music/              # Music files
    ├── sfx/                # Sound effects
    └── samples/            # Short samples
```

Part 11: Resources & Next Steps

Essential Resources

Documentation:

- 📚 Py5: https://py5coding.org
- 🔊 Processing Sound: https://processing.org/reference/libraries/sound/

Tools:

- 🎵 Audacity (free audio editor)
- 🎹 MuseScore (notation software)

Sound Libraries:

- 🎤 Freesound.org
- 🎵 Zapsplat.com

Next Steps

1. **Experiment** with the examples
2. **Combine** different analysis techniques
3. **Create** your own visualizers
4. **Add** interactivity (keyboard, mouse)
5. **Share** your projects!

Challenge: Build a music player with visualizer!

Advanced Topics to Explore

- 🎼 MIDI integration
- 🎛️ Custom DSP algorithms
- 🎵 Music theory integration
- 🤖 Machine learning on audio
- 🎪 Live performance tools
- 📊 Advanced signal processing

Key Takeaways

✅ Sound is data that can be analyzed and manipulated
✅ FFT reveals frequency content of audio
✅ Synthesis creates sound from scratch
✅ Effects transform audio characteristics
✅ Real-time processing enables interaction
✅ Visualization makes sound visible

The possibilities are endless! 🎵🎨

Summary - Component Ecosystem

Py5 + Sound Library

INPUT

- SoundFile
- AudioIn

ANALYSIS

- Amplitude
- FFT
- Waveform

SYNTHESIS

- Oscillators
- Noise

EFFECTS

- Filters
- Reverb/Delay

Thank You! 🎵

Questions?

Get Started:

```
pip install py5
python -c "import py5_tools; \
  py5_tools.processing.download_library('Sound')"
```

Happy Coding!

Appendix: Quick Reference

Common Methods:

```
# Playback
sound.play()
sound.loop()
sound.stop()
sound.amp(0.5)
sound.rate(1.0)
sound.pan(0.0)

# Analysis
amplitude.analyze()
fft.analyze()
fft.spectrum[i]
waveform.data[i]
```

Appendix: Frequency Chart

Frequency	Description
20-60 Hz	Sub-bass
60-250 Hz	Bass
250-500 Hz	Low midrange
500-2000 Hz	Midrange
2000-4000 Hz	Upper midrange
4000-6000 Hz	Presence
6000-20000 Hz	Brilliance

Appendix: Musical Note Reference

```
# Note frequencies (Hz)
notes = {
    'C4': 261.63, 'D4': 293.66, 'E4': 329.63,
    'F4': 349.23, 'G4': 392.00, 'A4': 440.00,
    'B4': 493.88, 'C5': 523.25
}
```

A4 = 440 Hz (universal tuning standard)

Appendix: About These Slides

Diagrams rendered using:

- Kroki.io for Mermaid diagrams
- HTML/CSS for layouts and comparisons
- ASCII art for simple flows