Chapter Objectives

The media learning goals for this chapter are:

- To blend sounds so that one fades into another.
- To create echoes.
- To change the frequency (pitch) of a sound.
- To create sounds that don’t exist in nature by composing more basic sounds (sine waves).
- To choose between sound formats such as MIDI and MP3 for different purposes.

The computer science goals for this chapter are:

- To use file paths to reference files at different places on the disk.
- To explain blending as an algorithm that crosses media boundaries.
- To build programs from multiple functions.
Making more complex sounds

- We know that natural sounds are often the combination of multiple sounds.
- Adding waves in physics or math is hard.
- In computer science, it's easy! Simply add the samples at the same index in the two waves:

```python
for srcSample in range(0, getLength(source)):
    destValue = getSampleValueAt(dest, srcSample)
    srcValue = getSampleValueAt(source, srcSample)
    setSampleValueAt(source, srcSample, srcValue + destValue)
```
Adding sounds

The first two are sine waves.

The third is just the sum of the first two columns.

\[ a + b = c \]
Uses for adding sounds

- We can mix sounds
  - We even know how to change the volumes of the two sounds, even over time (e.g., fading in or fading out)
- We can create echoes
- We can add sine (or other) waves together to create kinds of instruments/sounds that do not physically exist, but which sound interesting and complex
A function for adding two sounds

```python
def addSoundInto(sound1, sound2):
    for sampleNmr in range(0, getLength(sound1)):
        sample1 = getSampleValueAt(sound1, sampleNmr)
        sample2 = getSampleValueAt(sound2, sampleNmr)
        setSampleValueAt(sound2, sampleNmr, sample1 + sample2)
```

Notice that this adds sound1 and sound2 by adding sound1 *into* sound2
Making a chord by mixing three notes

```python
>>> c4=makeSound(getMediaPath("bassoon-c4.wav"))
>>> e4=makeSound(getMediaPath("bassoon-e4.wav"))
>>> g4=makeSound(getMediaPath("bassoon-g4.wav"))
>>> addSoundInto(e4,c4)
>>> play(c4)
>>> addSoundInto(g4,c4)
>>> play(c4)
```
def makeChord(sound1, sound2, sound3):
    for index in range(0, getLength(sound1)):
        s1Sample = getSampleValueAt(sound1, index)
        setSampleValueAt(sound1, index, s1Sample)
        if index > 1000:
            s2Sample = getSampleValueAt(sound2, index - 1000)
            setSampleValueAt(sound1, index, s1Sample + s2Sample)
        if index > 2000:
            s3Sample = getSampleValueAt(sound3, index - 2000)
            setSampleValueAt(sound1, index, s1Sample + s2Sample + s3Sample)

-Add in sound2 after 1000 samples
-Add in sound3 after 2000 samples

Note that in this version we’re adding into sound1!
Creating an echo

def echo(sndFile, delay):
    s1 = makeSound(sndFile)
    s2 = makeSound(sndFile)
    for index in range(delay, getLength(s1)):
        echo = 0.6*getSampleValueAt(s2, index-delay)
        combo = getSampleValueAt(s1, index) + echo
        setSampleValueAt(s1, index, combo)
    play(s1)
    return s1

This creates a delayed echo sound, multiplies it by 0.6 to make it fainter and then adds it into the original sound.
def double(source):
    len = getLength(source) / 2 + 1
    target = makeEmptySound(len)
    targetIndex = 0
    for sourceIndex in range(0, getLength(source), 2):
        value = getSampleValueAt(source, sourceIndex)
        setSampleValueAt(target, targetIndex, value)
        targetIndex = targetIndex + 1
    play(target)
    return target

Doubling the frequency

Why +1 here?

Here’s the piece that does the doubling
```python
def half(source):
    target = makeEmptySound(getLength(source) * 2)
    sourceIndex = 0
    for targetIndex in range(0, getLength(target)):
        value = getSampleValueAt(source, int(sourceIndex))
        setSampleValueAt(target, targetIndex, value)
        sourceIndex = sourceIndex + 0.5
    play(target)
    return target
```

This is how a sampling synthesizer works!

Here's the piece that does the halving
Can we generalize shifting a sound into other frequencies?

- **This way does NOT work:**

```python
def shift(source, factor):
    target = makeEmptySound(getLength(source))
    sourceIndex = 0

    for targetIndex in range(0, getLength(target)):
        value = getSampleValueAt( source, int(sourceIndex))
        setSampleValueAt( target, targetIndex, value)
        sourceIndex = sourceIndex + factor

    play(target)
    return target
```
Watching it not work

It’ll work for shifting down, but not shifting up. Why?

```python
>>> hello=pickAFile()
>>> print hello
/Users/guzdial/mediasources/hello.wav
>>> lowerhello=shift(hello,0.75)
>>> higherhello=shift(hello,1.5)
I wasn't able to do what you wanted.
The error java.lang.ArrayIndexOutOfBoundsException has occurred
Please check line 7 of /Users/guzdial/shift-broken.py
We need to prevent going past the end of the sound

def shift(source, factor):
    target = makeEmptySound(getLength(source))
    sourceIndex = 0

    for targetIndex in range(0, getLength(target)):
        value = getSampleValueAt(source, int(sourceIndex))
        setSampleValueAt(target, targetIndex, value)
        sourceIndex = sourceIndex + factor
        if sourceIndex > getLength(source):
            sourceIndex = 0

    play(target)
    return target
Sampling as an Algorithm

- Think about the similarities between:
  - Halving the sound’s frequency and scaling a picture larger.
  - Doubling the sound’s frequency and scaling a picture smaller.
def half(filename):
    source = makeSound(filename)
target = makeSound(filename)

    sourceIndex = 1
    for targetIndex in range(1, getLength(target)+1):
        setSampleValueAt(target, targetIndex, getSampleValueAt(source, int(sourceIndex))
    sourceIndex = sourceIndex + 0.5

    play(target)
    return target

def copyBarbsFaceLarger():
    # Set up the source and target pictures
    barbf=getMediaPath("barbara.jpg")
    barb = makePicture(barbf)
    canvasf = getMediaPath("7inX95in.jpg")
    canvas = makePicture(canvasf)
    # Now, do the actual copying
    sourceX = 45
    for targetX in range(100,100+((200-45)*2)):
        sourceY = 25
        for targetY in range(100,100+((200-25)*2)):
            color = getColor(getPixel(barb, int(sourceX), int(sourceY)))
            setColor(getPixel(canvas, targetX,targetY), color)
            sourceY = sourceY + 0.5
            sourceX = sourceX + 0.5
    show(barb)
    show(canvas)
    return canvas
Our programs (functions) implement algorithms

- Algorithms are descriptions of behavior for solving a problem.
- A program (our Python function) is an executable interpretations of algorithms.
- The same algorithm can be implemented in many different languages.
  - The same algorithm can be applied to many different data sets with similar results.
Both of these functions implement a sampling algorithm

- Both of them do very similar things:
  Get an index to a source
  Get an index to a target

For all the elements that we want to process:
  Copy an element from the source at the integer value of the source index
to the target at the target index
  Increment the source index by 1/2
Return the target when completed

This is a description of the algorithm.
Adding sine waves to make something completely new

- We saw earlier that complex sounds (like the sound of your voice or a trumpet) can be seen as being a sum of sine waves.
- We can *create* complex sounds by summing sine waves.
- These are sounds made by mathematics, by invention, not based on anything in nature.
Basic idea: Build a sine wave

- If we want a 440 Hz sound wave, then we need one of these cycles every \( \frac{1}{440} \)th of a second.
- We need to break this wave into the number of pieces in our sampling rate.

\[
\text{interval} = \frac{1}{\text{frequency}} \\
\text{samplesPerCycle} = \frac{\text{interval}}{1/\text{samplingRate}} = (\text{samplingRate})(\text{interval})
\]
Sound synthesis techniques

- Adding sine and square (and triangle) waves is additive sound synthesis.
- Most common modern synthesis technique is frequency modulation (FM) synthesis.
  - Much richer sound.
- Just about any way you can imagine to fill a sound mathematically can lead to an interesting synthesis technique.
  - Create random noise, then filter parts out: Subtractive synthesis
Adding envelopes

• Most real synthesizers today also allow you to manipulate envelopes
  • An envelope is a definition of how quickly the aspects of the sound change over time
  • For example, the rise in volume (attack), how the volume is sustained over time (sustain), how quickly the sound decays (decay): The ASD envelope
• Pianos tend to attack quickly, then decay quickly (without pedals)
• Flutes tend to attack slowly and sustain as long as you want.
Why write sound programs?

• “Aren’t there audio tools that can do many of these things?”
• Sure, and that’s good enough...if that’s good enough.
  • If you just want to use a sound, then simply using tools to generate the noise/instrument/sound you want is fine.
Communicating process

- What if you want to tell someone else how you got that sound, so that they can replicate the process, or even modify the sound in some way, or make it better?
- You could write down all the steps in a sound application tool.
  - Tedious, error prone.
- Or you could provide a program.
  - A succinct, executable definition of a process.
What is MP3?

- MP3 files are files encoded according to the MPEG-3 standard.
- They are audio files, but they are compressed in special ways.
  - They use a model of how we hear to get rid of some of the sound.
    - If there is a soft sound at the same time as a loud sound, don’t record the soft sound
  - They use various compression techniques to make the sound smaller.
- WAV files are compressed, but not as much, and don’t use any smart models to make themselves smaller.