Chapter 6: Modifying Sounds Using Loops
How sound works:
Acoustics, the physics of sound

- Sounds are waves of air pressure
  - An increase in pressure is a compression
  - A decrease in pressure is an rarefaction
- Compressions and rarefactions allow us to hear sound
- Our perception of sound is influenced by:
  - Shape of the wave
  - Amplitude
  - Frequency
How sound works:
Acoustics, the physics of sound

- The simplest sound is a sine wave
  - Compressions and rarefactions are of the same size and regularity
  - One compression plus one rarefaction is called a cycle
The physics of sound

- The **amplitude** is the maximum height of the wave, on either side
  - Amplitude is the most important factor in our perception of **volume**
- The **frequency** of a wave is the number of cycles per second (cps), or **Hertz**
  - Complex sounds have more than one frequency in them.
Volume and Pitch:
Psychoacoustics, the psychology of sound

- Our perception of volume is related (logarithmically) to changes in amplitude
  - If the amplitude doubles, it’s about a 3 decibel (dB) change
- Our perception of pitch is related (logarithmically) to changes in frequency
  - Higher frequencies are perceived as higher pitches
  - We can hear between 20 Hz and 20,000 Hz (20 kHz)
  - Example: in a piano, A above middle C is 440 Hz
“Logarithmically?”

- Our hearing works on **ratios** not **absolute differences**, for example:
  - We hear the difference between 200 Hz and 400 Hz, as the same as 500 Hz and 1000 Hz
  - Similarly, 200 Hz to 600 Hz, and 1000 Hz to 3000 Hz
Decibel is a logarithmic measure

- The change of intensity (volume) of a sound is measured in Decibels (dB)
- A decibel is a ratio between two intensities: $10 \times \log_{10}(I_1/I_2)$
- When decibels are used as an absolute measure, it’s in comparison to threshold of audibility
- Zero dB can’t be heard.
- Normal speech is 60 dB.
- A shout is about 80 dB
A few more ideas about sounds

- Real sounds are almost never single-frequency sound waves
  - Most natural sounds have several frequencies in them, often at different amplitudes
    - Overtones → additional frequencies
- Not all sound waves are well represented by sine waves
  - Real sounds have funny bumps and sharp edges
Encoding Sound

- To digitize sound means to take a flow of waves and turn it into numbers
  - Measure the amount of air pressure as a single number
    - Positive numbers $\rightarrow$ compressions
    - Negative numbers $\rightarrow$ rarefactions
    - ADC: Analog to Digital Conversion
  - Each pressure reading is a **sample**
    - Q: How many samples do we need to have a good sound representation?
    - A: it depends on the highest frequency of the sound
Nyquist Theorem

- We need twice as many samples as the maximum frequency in order to represent (and recreate, later) the original sound.
  - If we capture 8000 samples per second, the highest frequency we can capture is 4000 Hz
  - If we capture more than 44,000 samples per second, we capture everything that we can hear (max 22,000 Hz)

- The number of samples collected per second is the **sampling rate**
  - CD quality is 44,100 samples per second
Digitizing sound in the computer

- Each sample is stored as a number, using two bytes (or 16 bits)
- What’s the range of available combinations?
  - 16 bits, \(2^{16} = 65,536\)
  - But we want both positive and negative values
    - To indicate compressions and rarefactions.
  - We will use one bit to indicate positive (0) or negative (1)
  - That leaves us with 15 bits
  - 15 bits, \(2^{15} = 32,768\)
  - One of those combinations will stand for zero
    - We’ll use a “positive” one, so that’s one less pattern for positives
  - Each sample then can be a value between \([-32,768, 32,767]\)
Two’s Complement Numbers

- The way computers represent positive and negative numbers
- Positive numbers are shown as usual in binary
  - Example: 9 in binary is \(0001001\)
  - How about -9??
- Negative numbers are calculated and represented as follows:
  - Start with the positive number in binary: \(0001001\)
  - Invert all the digits: \(1110110\)
  - Add 1 to the result: \(1110110 + 1 = 11110111\)
Sounds as arrays

- Samples are just stored one right after the other in the computer’s memory.
  
  (Like pixels in a picture)

- That’s called an **array**
  
  - It’s an especially efficient (quickly accessed) memory structure

<table>
<thead>
<tr>
<th></th>
<th>59</th>
<th>39</th>
<th>16</th>
<th>10</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Sounds as arrays

- Each value in the array is known as an element.
- We track each element in the array with an integer number, called index.
- The first element in the array is at index 0.
- The last element in the array is at an index equal to the number of elements in the array minus 1 (\textit{size} – 1).

\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
59 & 39 & 16 & 10 & -1 \\
\hline
0 & 1 & 2 & 3 & 4 \\
\hline
\end{tabular}
\end{center}
Getting elements of an array

- To access an element of an array we use the subscript operator `[]` and a valid index
- For example, in the following array (named `A`):

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>39</td>
<td>16</td>
<td>10</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

```python
>>> print A[0]
59
-1
```
Working with sounds

- Encodings (types) we will be using:
  - **Sound**: encoding of sound (from a wav file)
  - **Samples**: an array of samples
  - **Sample**: a sound sample with value in the range -32768 to 32767
Working with sounds

- We’ll use `pickAFile()` and `makeSound(filename)` to create sounds
  - `.wav` files
- We’ll use `play` and `blockingPlay` to play sounds
- We can see the samples of a sound by using `explore`
- We’ll use `getSamples(sound)` to get all the sample objects out of a sound
- Sounds know their length (`getLength`) and their sampling rate (`getSamplingRate`)
Working with Samples

- We can get the **sample value** at any index with `getSampleValueAt(sound, index)`
- We can change a sample with `setSampleValueAt(sound, index, value)`
- A sample object remembers its sound, so if you change the sample object, the sound gets changed.
- Sample objects also understand `getSampleValue(sample)` and `setSampleValue(sample, value)`
- We can save sounds to the hard-drive with `writeSoundTo(sound, "fileName.wav")`
But there are thousands of these samples!

- How do we do something to these samples to manipulate them, when there are thousands of them per second?
- We use a loop and get the computer to iterate in order to do something to each sample.
- An example loop:

  ```
  for sample in getSamples(sound):
    value = getSampleValue(sample)
    setSampleValue(sample, value * 2)
  ```

- Example: `increaseVolume()`
How did that work?

- When we evaluate `increaseVolume(s)`, the function `increaseVolume` is executed.
  - The sound in the variable `s` becomes known as `sound`.
  - `sound` is a placeholder (a.k.a parameter) for the sound object `s`.

```python
>>> f = pickAFile()
>>> s = makeSound(f)
>>> increaseVolume(s)
```
We can make this generic

- By adding another *parameter*, we can create a general *changeVolume* that can increase or decrease volume.
  - A value greater than 1 will increase the volume
  - A value less than 1 will decrease the volume

```python
def changeVolume(sound, factor):
    for sample in getSamples(sound):
        value = getSampleValue(sample)
        setSampleValue(sample, value * factor)
```
Normalizing sounds: Maximizing volume

- How do we get maximum volume?
- It’s a three-step process:
  - First, figure out the loudest piece of the sound (largest sample).
  - Next, figure out how much we have to increase/decrease that sample to fill the available space (32767)
    - We want to find the amplification factor \( amp \), where \( amp \times \text{loudest} = 32767 \)
    - In other words: \( amp = \frac{32767}{\text{loudest}} \)
  - Finally, amplify each sample by multiplying it by \( amp \)
Python Function: max()

- **max()** is a function that takes *any* number of inputs, and always returns the largest.
- There is also a function **min()** which works similarly but returns the minimum.
- **Example:** `normalize(sound)`

```python
>>> print max(1,2,3)
3
>>> print max(4,67,98,-1,2)
98
>>> print min(1,2,3)
1
>>> print min(4,67,98,-1,2)
>>>-1
```
Avoiding clipping

- Clipping: the effect that occurs when the normal curves of the sound are broken (clipped) by the limitations of the sample size
- Why are we being so careful to stay within range? What if we just multiplied all the samples by some big number and let some of them go over 32,767? The result then is *clipping*
  - The awful, buzzing noise whenever the sound volume is beyond the maximum that your sound system can handle.
- **Example:** `clipping(sound)`
Processing only part of the sound

- What if we wanted to increase or decrease the volume of only part of the sound?
- Q: How would we do it?
- A: We’d have to use a `range()` function with our for loop
  - Just like when we manipulated only part of a picture by using `range()` in conjunction with `getPixels()`
  - And we use the `for` loop like we did with pictures

```python
for index in range (0, getLength(sound) / 2) :
```