Volume

As we have learned, the amplitude of a sound is the main factor in the volume. If the amplitude of a sound increases or decreases, the volume of a sound increases and decreases accordingly. To change the volume of a sound, we need to change the values of the amplitude that are stored in each sample.

Working with samples in a sound

We have seen that to get a sound to work with, we will use a command such as:

$$sound = makeSound(pickAFile(\ ))$$

Just as we used the `getAllPixels` function to get a list of all of the pixels in a picture, we can use the `getSamples` function to get a list of all of the samples in a sound. If we want to get the value of a particular sample object, we can use the `getSampleValue` function, which takes a sample object as input, and returns the value of that sample (*i.e.*, the measure of the amplitude at that instance). If we would like to change the value of the sample, we can use the `setSampleValue` function, which takes a sample object and a value (between -32768 and 32767) as parameters, and sets the value of the sample to be the specified value.

For example, suppose we want to increase the volume in a sound. This means the amplitude of the sound will need to be increased. We can do this in code with the following function:

Example: Increase Volume

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

  return newSound
```

Let's look at what is going on in this function. We pass a sound object into the function, and then the very first thing we do is duplicate that sound, so that we make our changes to the copy of the sound. Doing this ensures that we do not modify the original sound. We then loop through all of the samples in the new sound, getting the sample value of each, and then changing the value of the sample to be twice as large.

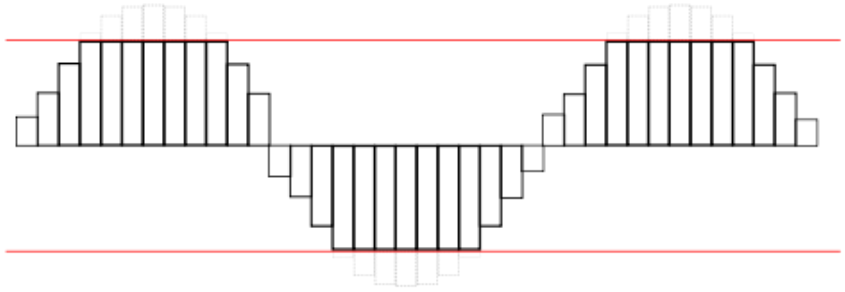
How do we know this function worked? When we play the original sound, and then the new sound, we can probably hear that the new sound is louder. We can also

open both sounds using the `soundGraph` function provided in the Changing Volume min-lab. We should find that the values in the new sound are double those in the original.

Another way to check the value of samples is to use the `getSampleValueAt` function. This function takes a sound and an index as parameters and returns the value of the sample at that index in the sound array. We could compare sample values of the original sound to the new sound by printing particular values, such as in these examples:

```
>>> print getSampleValueAt(s, 0)
-315
>>> print getSampleValueAt(newS, 0)
-630
>>> print getSampleValueAt(s, 16730)
-16773
>>> print getSampleValueAt(newS, 16730)
-32768
```

What happened with the sample value at index 16730? The original sample value was -16773, but when this was multiplied by 2, it should have been -33546. Remember, because we are using 16-bit two's complement representation for the storage of sample values, the values must range between -32768 and 32767. So this new sample value has been capped at -32768. This is what's known as *clipping*. The amplitude of the sound gets stopped at the maximum capacity. When this happened, the sine wave that represents the sound looks squared-off at the peaks, similar to:



We may also loop through the sound using the range function and the `getNumSamples` function. The `getNumSamples` function takes a sound as input and returns the number of samples in the sound. (Alternatively, the `getLength`

function also returns the number of samples in a sound.) The increaseVolume function would be modified as follows:

Example: Increase volume, version 2

```
def increaseVolume(sound):
  newSound = duplicateSound(sound)
  for index in range(getNumSamples(newSound)):
    value = getSampleValueAt(newSound, index)
    setSampleValueAt(newSound, index, value * 2)

  return newSound
```

This method of looping is similar to how we looped over x- and y-values in pictures, and used those values to get pixels to work with. Here, we use the index to get sample values to work with.

We should now take some time to experiment with changing volume, by working through the Mini-Lab: Changing Volume.

Note for future mini-lab or exercises– it would be better to have them write functions such as changeVolume and increaseAndDecrease and then fade.