## Sound Manipulations

In the previous section, we discussed how to change the volume of a sound by modifying the values of the amplitude stored in each sample. We used a loop to go through each sample of the sound, assigning them new values. This is similar to using loops to go through each pixel in an image and assigning them new red, green, and blue values. Once we learned how to loop through all of the pixels in an image, we were then able to modify the ranges to go through subsets of the pixels. We also added conditional statements to be selective about which pixels we wanted to change, as well as how to use the color values of pixels in one image to set the color values of pixels in another image. By using different ranges of samples, we can modify sections of a sound instead of the entire sound (like in the `increaseAndDecrease` function from the previous activity), and by using conditional statements, we can be selective about which samples to modify. We will also see that we can use the sample values in one sound to set the sample values in another sound. In this reading we will explore several sound manipulations where we use modified ranges and/or get sample values from one sound and put them into another sound.

The first example we will explore is reversing a sound. Recall, a sound is stored as an array of sample values. If you think of this as an ordered list of values, then reversing the sound would mean putting this list in reverse order. To do this, we will create a new sound with the same size as the original sound, and then as we get the sample values from the original sound, we will set them at the opposite end in the new sound. If a sound has `n` samples, then we have the following correspondence between sample indices in the original sound and sample indices in the new sound:

| Index in original sound | Index in new sound |
|---|---|
| 0 | n-1 |
| 1 | n-2 (*i.e.*, n-1-1) |
| 2 | n-3 (*i.e.*, n-1-2) |
| … | |
| n-1 | 0 (*i.e.*, n-1-(n-1)) |

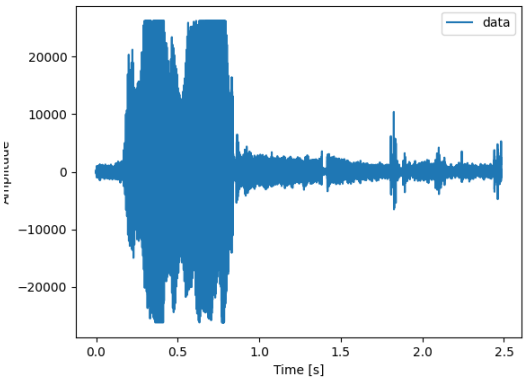
The code is shown below:

**Example:** Reversing a sound

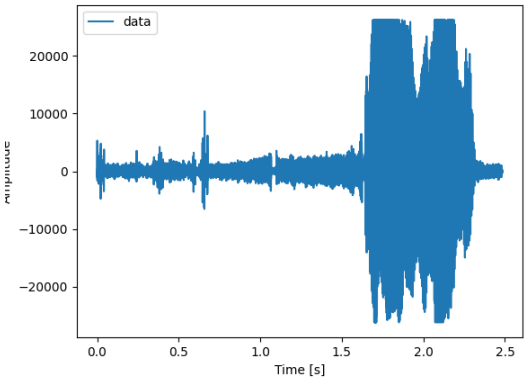
```
# function to reverse a sound array
def reverse(soundData):
  # Make an array of all zeros, same length as soundData,
  # with data type np.int16
  new_data = np.zeros(len(soundData), dtype=np.int16)
```

```
    numSamples = len(soundData)

# loop through soundData, setting the sample values in new_data
    for index in range(numSamples):
      new_data[numSamples -1 - index] = soundData[index]

    return new_data
```

The sound waves would look something like the following:



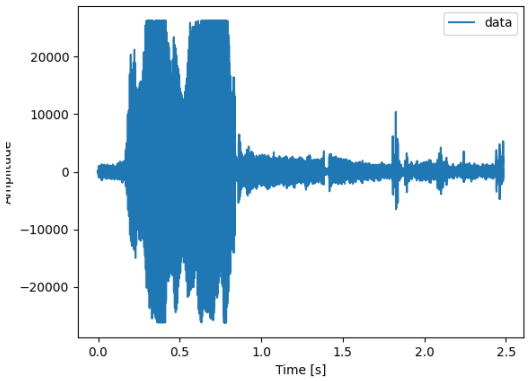

original sound



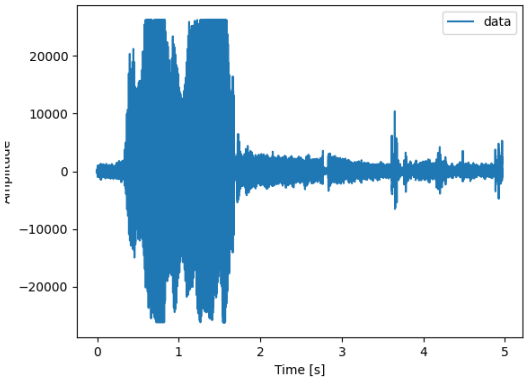

reversed sound

In our introductory discussion about sound, we learned that pitch is related to frequency. We can change the pitch of a sound by changing the frequency. Sound frequency is determined by the way in which sound waves oscillate while travelling to our ears; they alternate between compressing and stretching the medium. The frequency of a sound wave is measured as the total number of waves produced in one second.

In the following figures of sound waves, we can see that the original sound is being "stretched", or more precisely, the frequency has been halved. (In the original length of time of the sound, we now have only half the sound.) Each sample in the original sound is being used twice in the new sound. So, the length of time for one cycle has been doubled. When you look at the waves for the sounds in these pictures, you can see that the sounds have the same shape, but the second one looks stretched. (Look at the Time axis along the bottom of the figure.)




original sound



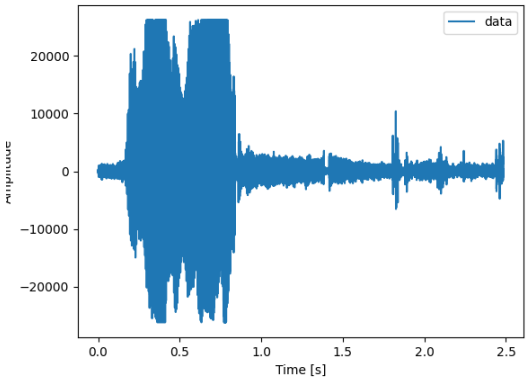

sound with halved frequency

So how do we do this with code? Our algorithm is very similar to what we did to scale a picture up in size. To make a picture larger, we used every pixel twice. We are going to do the same thing with samples in our sounds – we will use each sample value twice to stretch the sound into a new sound twice as long.

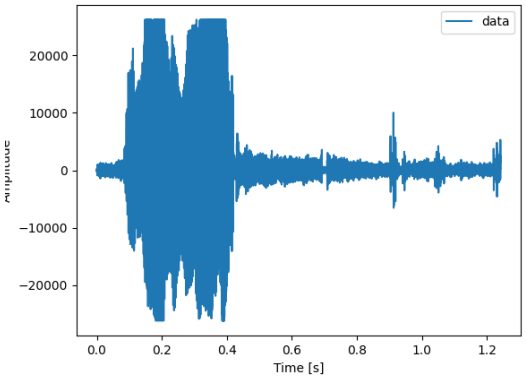
**Example:** Halving the frequency

```
# function to halve the frequency
# new sound is twice as long, and sounds lower and deeper
def halfFrequency(soundData):
  # Make an array of all zeros, twice as long as soundData,
  # data type np.int16
  new_data = np.zeros(len(soundData)*2, dtype=np.int16)

  # loop through soundData, setting sample values in new_data
  for index in range(len(new_data)):
    new_data[index] = soundData[index//2]

  return new_data
```

We could just as easily double the frequency of a sound. Again, we would use ideas similar to what we used to make a picture smaller. To make a picture smaller, we used every other pixel; to double the frequency, we will use every other sample value. So our original sound would look like this with a doubled frequency:




original sound




sound with doubled frequency

Again, notice the Time axis along the bottom is different between the two.

In the next activity, you will experiment with writing and testing functions to reverse a sound and to change the pitch of a sound by halving and doubling the frequency.

**Activity: Sound Manipulations**