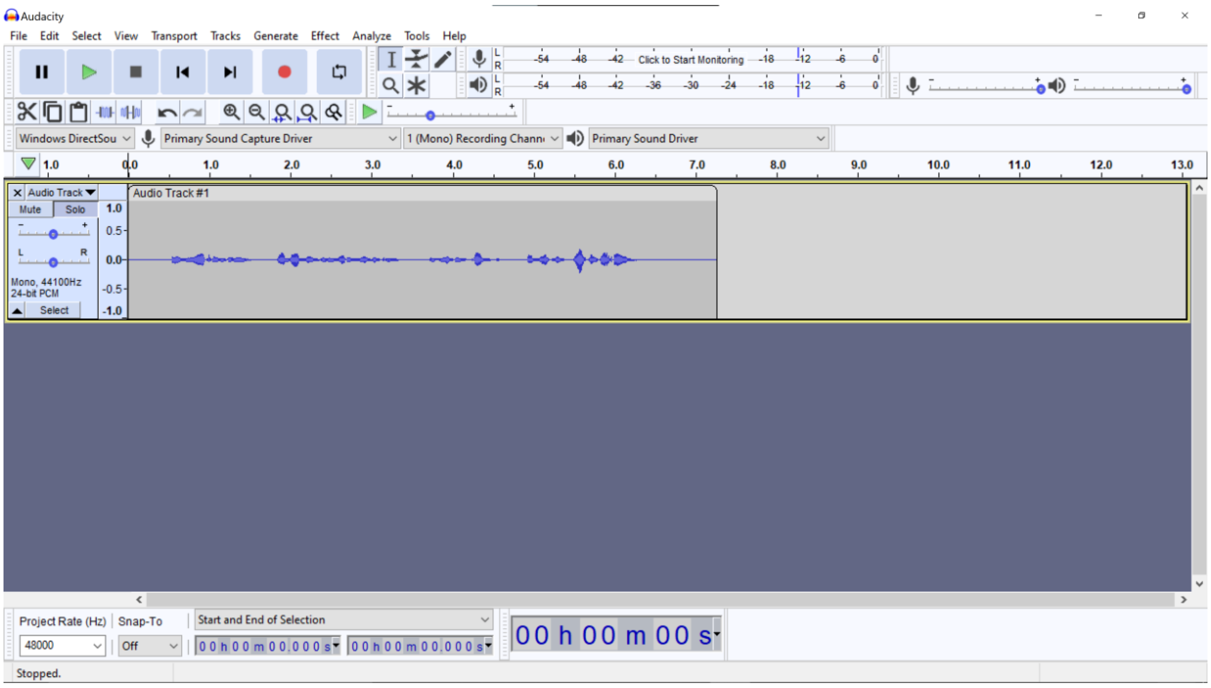
Editing in Audacity

For the recording process, I recorded my voice with monitoring on to ensure that my voice does not go below -1.0db, such that clipping will no occur. The amplitude did not exceed -6.0dB. The first screenshot shows the original recording without editing. Project Rate has been changed to 48000Hz and the recording is being saved in 24-bits.


Audacity
File Edit Select View Transport Tracks Generate Effect Analyze Tools Help
Click to Start Monitoring
Windows DirectSou
Primary Sound Capture Driver
1 (Mono) Recording Chann
Primary Sound Driver
Audio Track
Mute Solo
Select
Audio Track #1
Start and End of Selection

Mono, 44100Hz
24-bit PCM

Project Rate (Hz) 48000

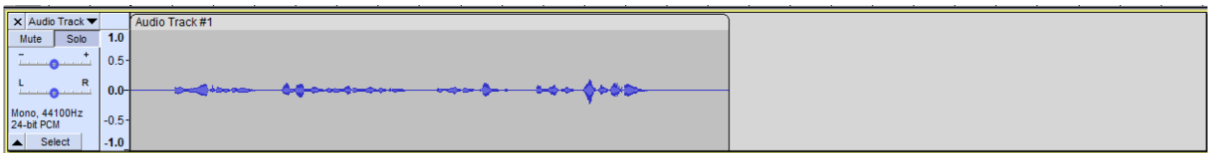
Snap-To Off

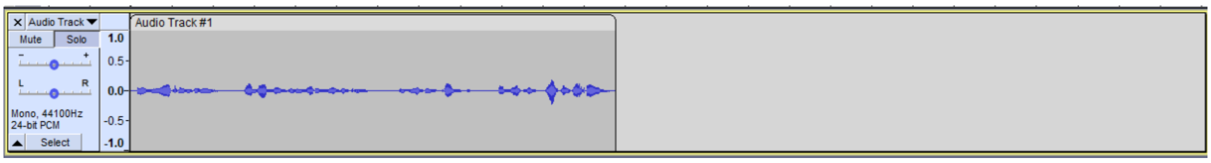
00 h 00 m 00.000 s 00 h 00 m 00.000 s

00 h 00 m 00 s

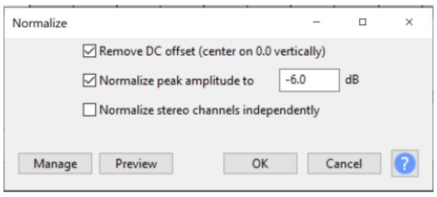
Stopped.

The two screenshots below show the before and after when I cut and remove the silences at the beginning and ending of the file.





To normalize, I then set the peak amplitude to -6.0dB. The screenshot below shows the process and results of it. Setting a level of -6dB just below the maximum amplitude is a small and since my recording amplitude did not exceed that during my monitoring process, I can ensure that there was no clipping.

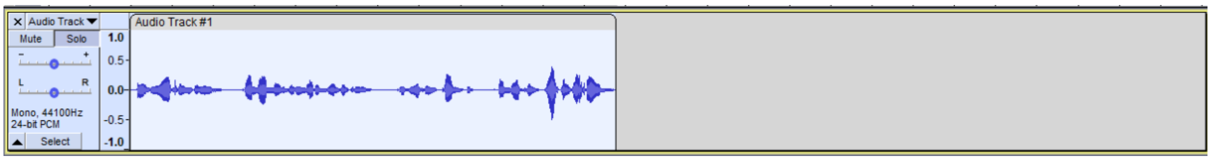


Normalize

☑ Remove DC offset (center on 0.0 vertically)

☑ Normalize peak amplitude to -6.0 dB

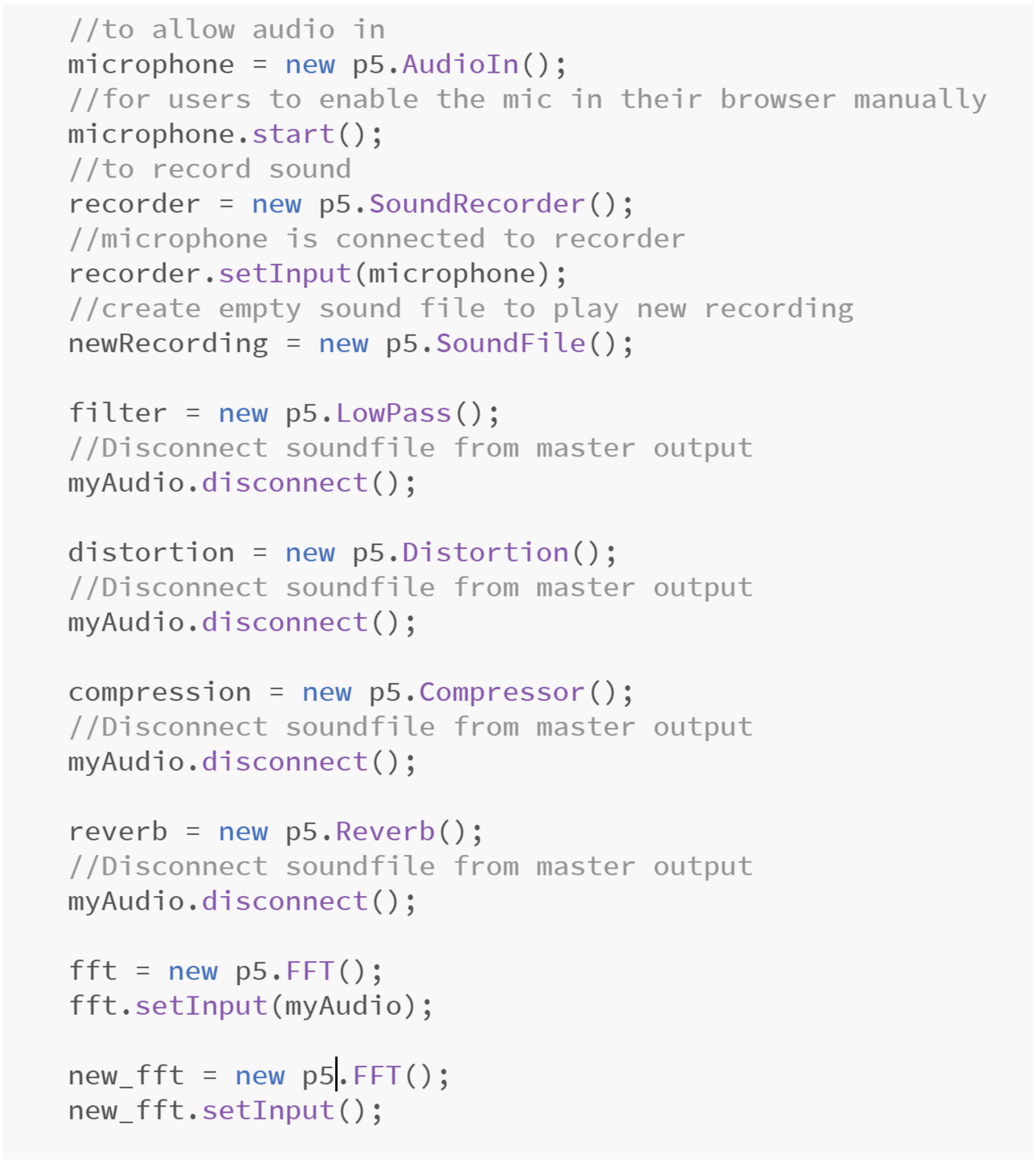
☐ Normalize stereo channels independently



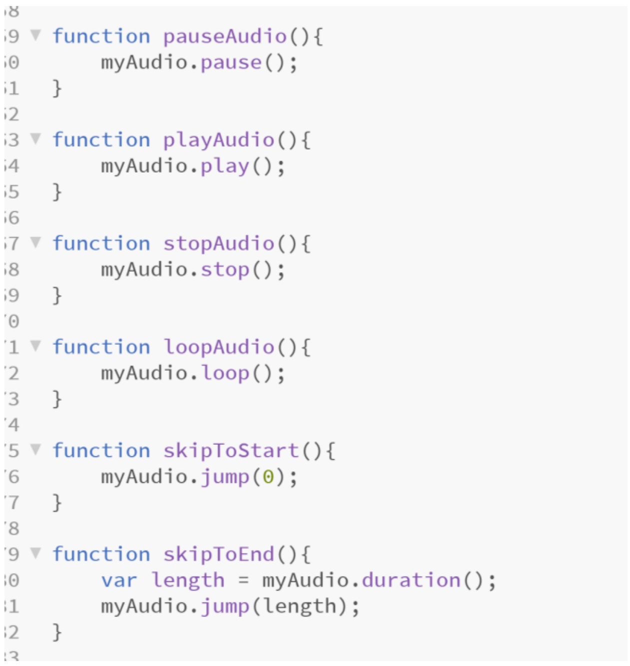
The recording is then saved in WAV format.

Main characteristics explained and analysis on how lowpass and master volume affects sound spectrum

Firstly, the objects for each feature is created in the setup() function to be called after.

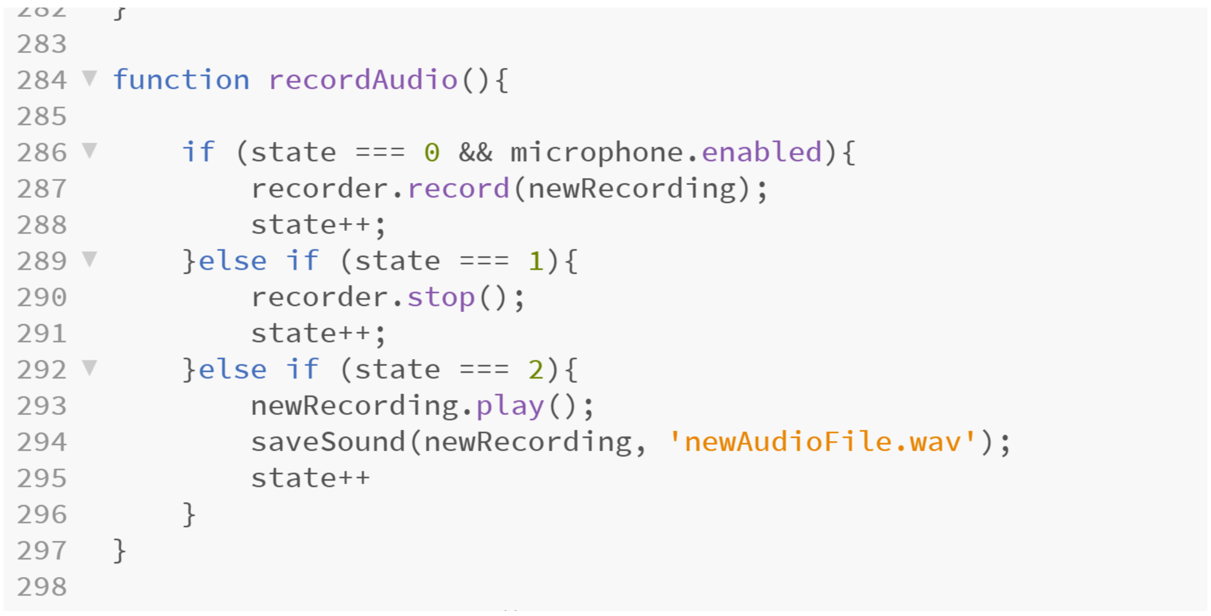


```
    //to allow audio in
    microphone = new p5.AudioIn();
    //for users to enable the mic in their browser manually
    microphone.start();
    //to record sound
    recorder = new p5.SoundRecorder();
    //microphone is connected to recorder
    recorder.setInput(microphone);
    //create empty sound file to play new recording
    newRecording = new p5.SoundFile();

    filter = new p5.LowPass();
    //Disconnect soundfile from master output
    myAudio.disconnect();

    distortion = new p5.Distortion();
    //Disconnect soundfile from master output
    myAudio.disconnect();

    compression = new p5.Compressor();
    //Disconnect soundfile from master output
    myAudio.disconnect();

    reverb = new p5.Reverb();
    //Disconnect soundfile from master output
    myAudio.disconnect();

    fft = new p5.FFT();
    fft.setInput(myAudio);

    new_fft = new p5.FFT();
    new_fft.setInput();
```



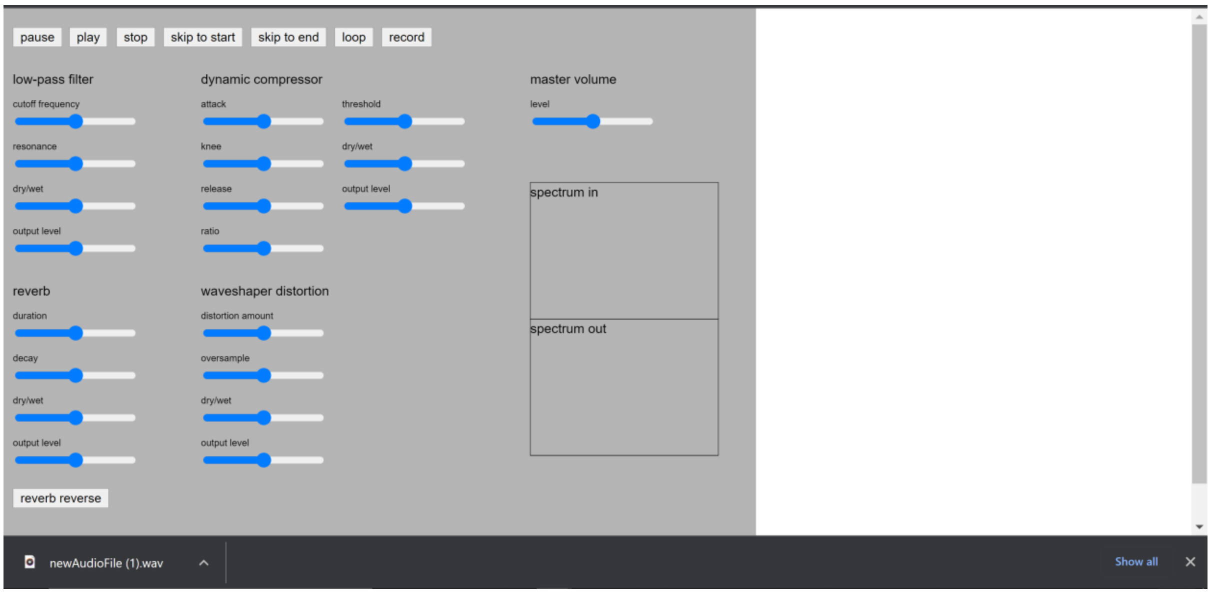
```
function pauseAudio(){
    myAudio.pause();
}

function playAudio(){
    myAudio.play();
}

function stopAudio(){
    myAudio.stop();
}

function loopAudio(){
    myAudio.loop();
}

function skipToStart(){
    myAudio.jump(0);
}

function skipToEnd(){
    var length = myAudio.duration();
    myAudio.jump(length);
}
```

The gui configuration function calls these functions above to enable the audio to play, stop, loop, restart, and skip to the end.



```

function recordAudio(){

    if (state === 0 && microphone.enabled){
        recorder.record(newRecording);
        state++;
    }else if (state === 1){
        recorder.stop();
        state++;
    }else if (state === 2){
        newRecording.play();
        saveSound(newRecording, 'newAudioFile.wav');
        state++
    }
}

```

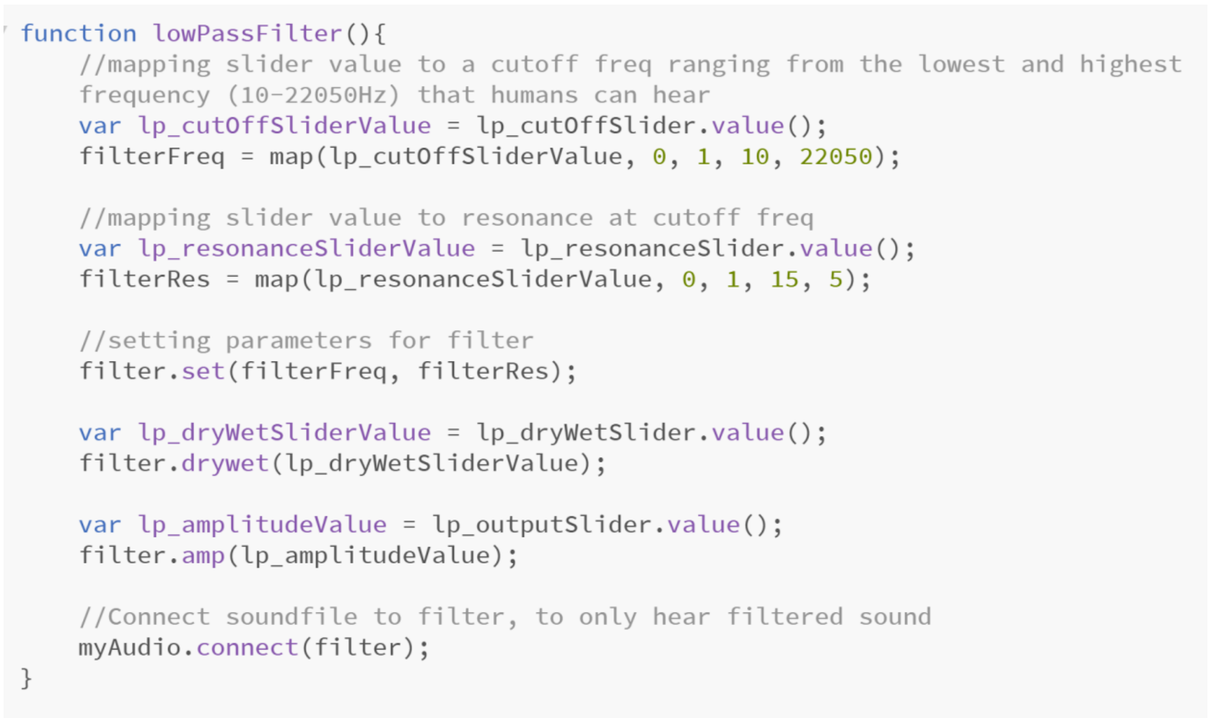
This shows how the recording is saved. The state increments as it represents the number of time the recording button is clicked and what it does each time.



My screen recording was unable to show the file explorer popping up when voice is being recorded, hence the screenshot below shows the newAudioFile.wav being recorded.

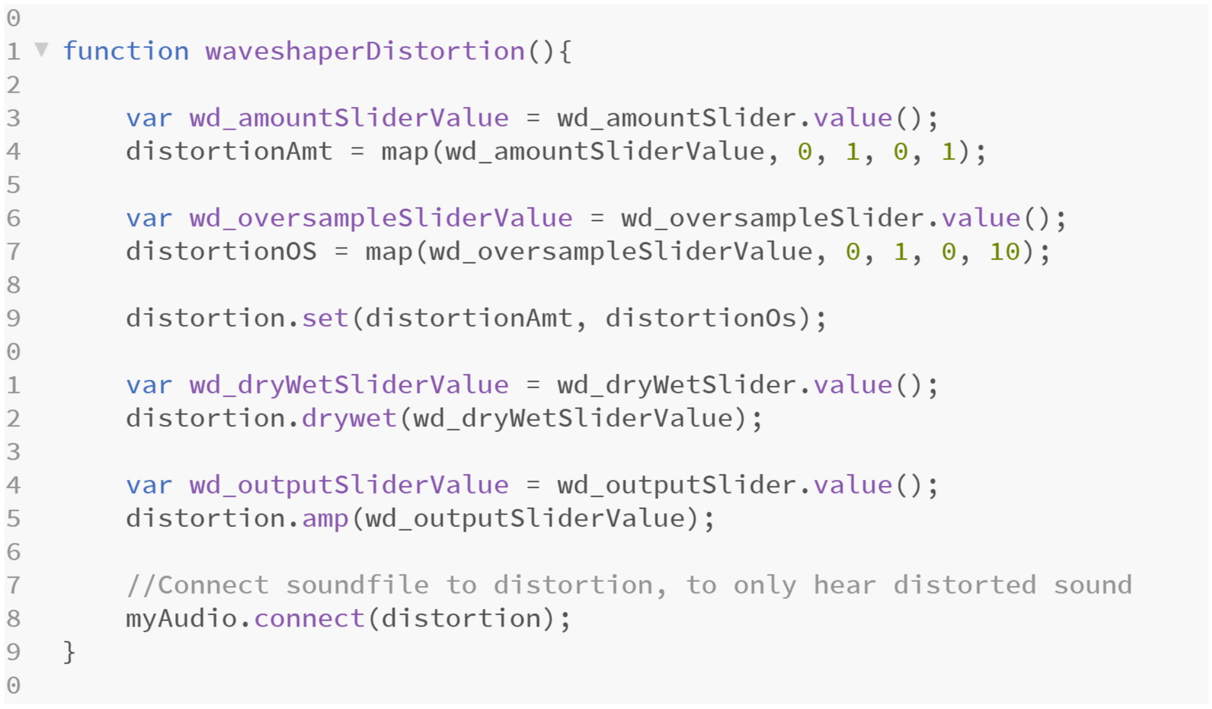
The low pass filter accentuates content above the cutoff frequency specified by the slider such that the filter allows lower frequencies to pass. This reduces high pitched noise and the spectrum will affected to display the lower frequencies more.

filterFreq maps the slider value from 0 to 1, to the corresponding filter frequency of 10 to 22050Hz. This is then set by the filter object in setup(), where the object calls on lowpass filter from p5. The process repeats for the other features.

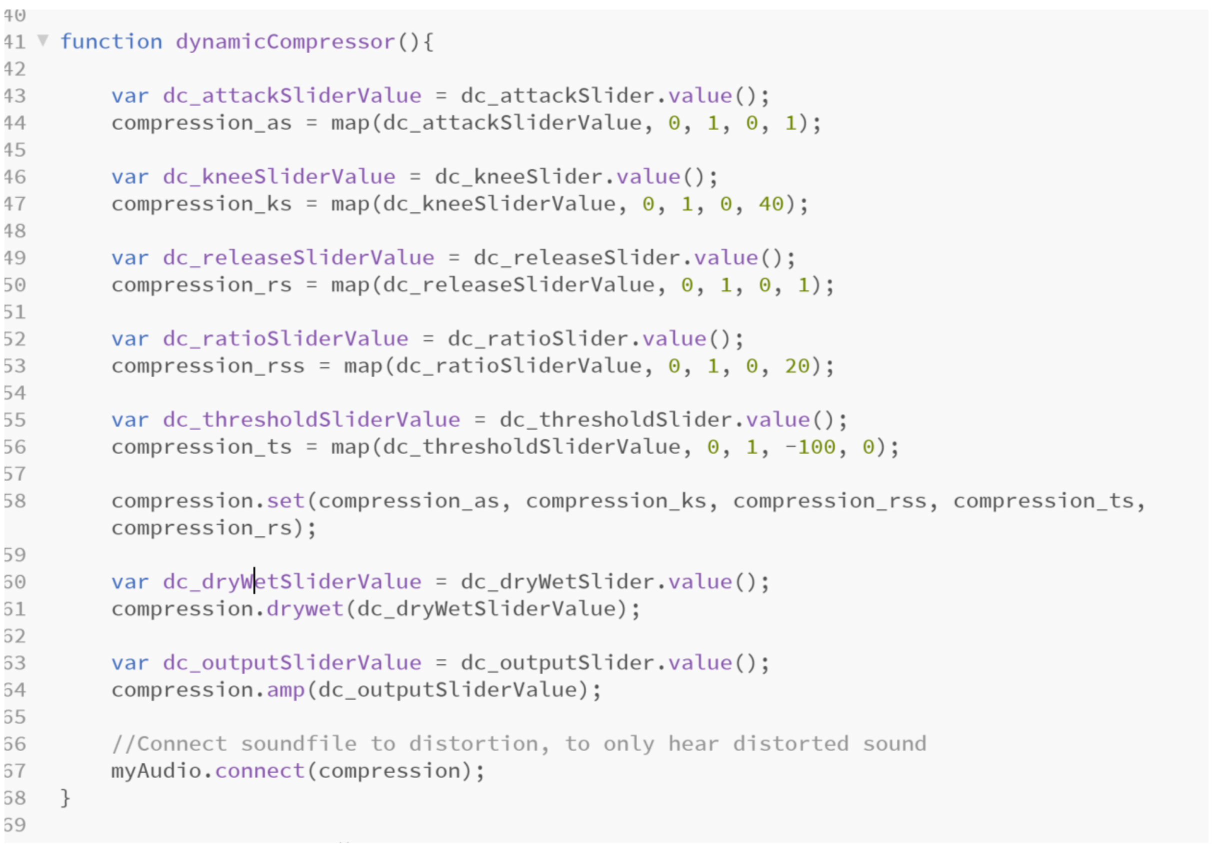


```
function lowPassFilter(){
    //mapping slider value to a cutoff freq ranging from the lowest and highest
    frequency (10-22050Hz) that humans can hear
    var lp_cutOffSliderValue = lp_cutOffSlider.value();
    filterFreq = map(lp_cutOffSliderValue, 0, 1, 10, 22050);

    //mapping slider value to resonance at cutoff freq
    var lp_resonanceSliderValue = lp_resonanceSlider.value();
    filterRes = map(lp_resonanceSliderValue, 0, 1, 15, 5);

    //setting parameters for filter
    filter.set(filterFreq, filterRes);

    var lp_dryWetSliderValue = lp_dryWetSlider.value();
    filter.drywet(lp_dryWetSliderValue);

    var lp_amplitudeValue = lp_outputSlider.value();
    filter.amp(lp_amplitudeValue);

    //Connect soundfile to filter, to only hear filtered sound
    myAudio.connect(filter);
}
```

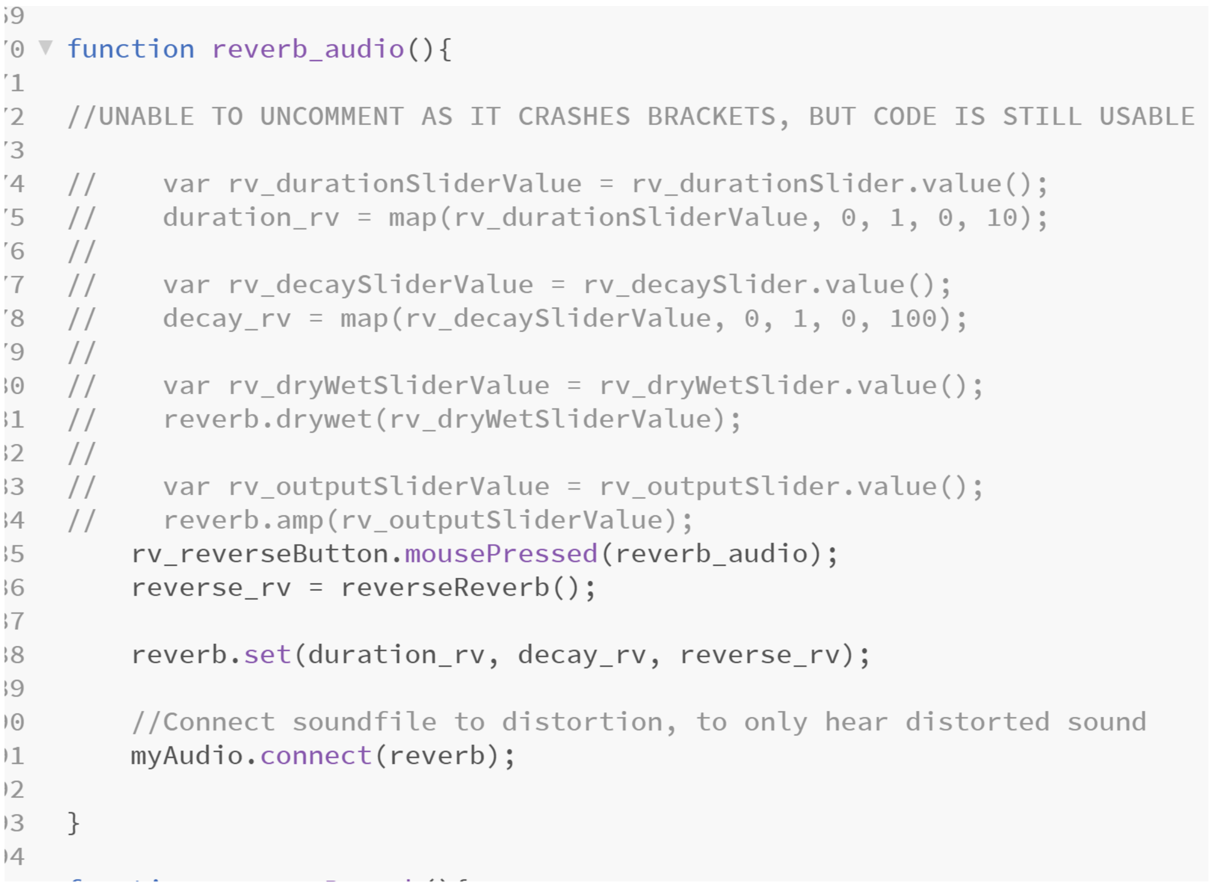
The previous process is the same for wavershaper distortion, dynamic compressor and reverb, where the corresponding object from p5 allows the features to be set and mapped from the slider input values to the sound that is being proceesed.



```
function waveshaperDistortion(){

    var wd_amountSliderValue = wd_amountSlider.value();
    distortionAmt = map(wd_amountSliderValue, 0, 1, 0, 1);

    var wd_oversampleSliderValue = wd_oversampleSlider.value();
    distortionOS = map(wd_oversampleSliderValue, 0, 1, 0, 10);

    distortion.set(distortionAmt, distortionOs);

    var wd_dryWetSliderValue = wd_dryWetSlider.value();
    distortion.drywet(wd_dryWetSliderValue);

    var wd_outputSliderValue = wd_outputSlider.value();
    distortion.amp(wd_outputSliderValue);

    //Connect soundfile to distortion, to only hear distorted sound
    myAudio.connect(distortion);
}
```

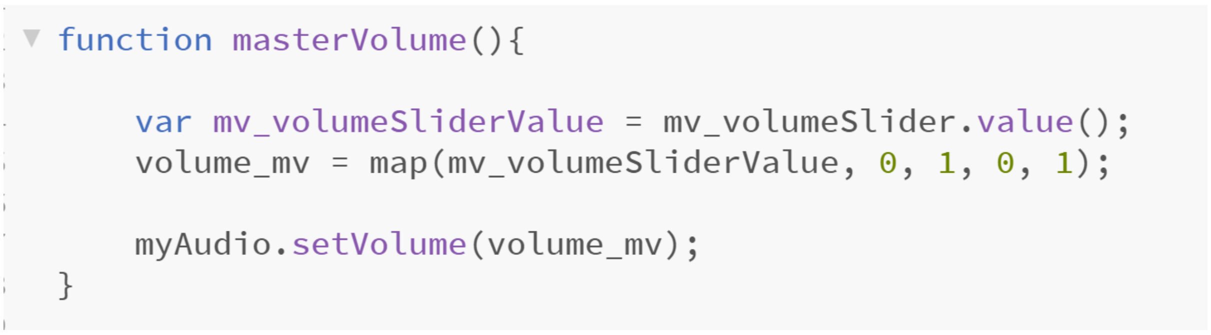


```
function dynamicCompressor(){

    var dc_attackSliderValue = dc_attackSlider.value();
    compression_as = map(dc_attackSliderValue, 0, 1, 0, 1);

    var dc_kneeSliderValue = dc_kneeSlider.value();
    compression_ks = map(dc_kneeSliderValue, 0, 1, 0, 40);

    var dc_releaseSliderValue = dc_releaseSlider.value();
    compression_rs = map(dc_releaseSliderValue, 0, 1, 0, 1);

    var dc_ratioSliderValue = dc_ratioSlider.value();
    compression_rss = map(dc_ratioSliderValue, 0, 1, 0, 20);

    var dc_thresholdSliderValue = dc_thresholdSlider.value();
    compression_ts = map(dc_thresholdSliderValue, 0, 1, -100, 0);

    compression.set(compression_as, compression_ks, compression_rss, compression_ts,
    compression_rs);

    var dc_dryWetSliderValue = dc_dryWetSlider.value();
    compression.drywet(dc_dryWetSliderValue);

    var dc_outputSliderValue = dc_outputSlider.value();
    compression.amp(dc_outputSliderValue);

    //Connect soundfile to distortion, to only hear distorted sound
    myAudio.connect(compression);
}
```



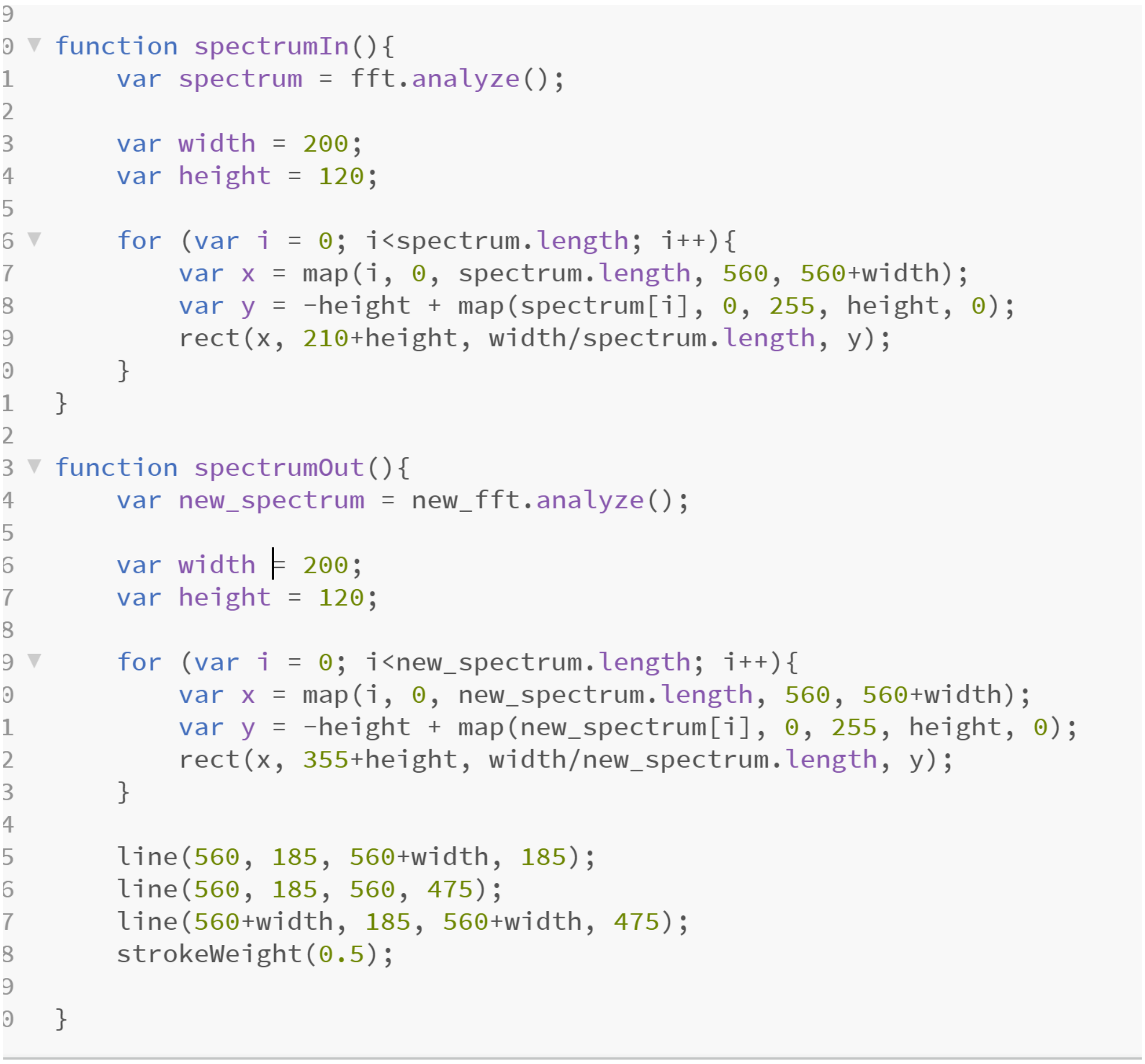
```
function reverb_audio(){

//UNABLE TO UNCOMMENT AS IT CRASHES BRACKETS, BUT CODE IS STILL USABLE

//    var rv_durationSliderValue = rv_durationSlider.value();
//    duration_rv = map(rv_durationSliderValue, 0, 1, 0, 10);
//
//    var rv_decaySliderValue = rv_decaySlider.value();
//    decay_rv = map(rv_decaySliderValue, 0, 1, 0, 100);
//
//    var rv_dryWetSliderValue = rv_dryWetSlider.value();
//    reverb.drywet(rv_dryWetSliderValue);
//
//    var rv_outputSliderValue = rv_outputSlider.value();
//    reverb.amp(rv_outputSliderValue);
    rv_reverseButton.mousePressed(reverb_audio);
    reverse_rv = reverseReverb();

    reverb.set(duration_rv, decay_rv, reverse_rv);

    //Connect soundfile to distortion, to only hear distorted sound
    myAudio.connect(reverb);

}
```

However, the reverb feature crashes my app brackets every time it is being run, hence the code is commented out to ensure that the site is function properly.



```
function masterVolume(){

    var mv_volumeSliderValue = mv_volumeSlider.value();
    volume_mv = map(mv_volumeSliderValue, 0, 1, 0, 1);

    myAudio.setVolume(volume_mv);
}
```

The master volume multiplies the combined amplitude or output volume of each filter between 0 and 1. The latter will increase the frequencies of sound to show the maximum amplitude, which will be reflected on the spectrum, and vice versa.



```

function spectrumIn(){
    var spectrum = fft.analyze();

    var width = 200;
    var height = 120;

    for (var i = 0; i<spectrum.length; i++){
        var x = map(i, 0, spectrum.length, 560, 560+width);
        var y = -height + map(spectrum[i], 0, 255, height, 0);
        rect(x, 210+height, width/spectrum.length, y);
    }
}

function spectrumOut(){
    var new_spectrum = new_fft.analyze();

    var width = 200;
    var height = 120;

    for (var i = 0; i<new_spectrum.length; i++){
        var x = map(i, 0, new_spectrum.length, 560, 560+width);
        var y = -height + map(new_spectrum[i], 0, 255, height, 0);
        rect(x, 355+height, width/new_spectrum.length, y);
    }

    line(560, 185, 560+width, 185);
    line(560, 185, 560, 475);
    line(560+width, 185, 560+width, 475);
    strokeWeight(0.5);

}
```

The function above shows how the spectrum analyze() and fft object is being used to draw the respective spectrums.