

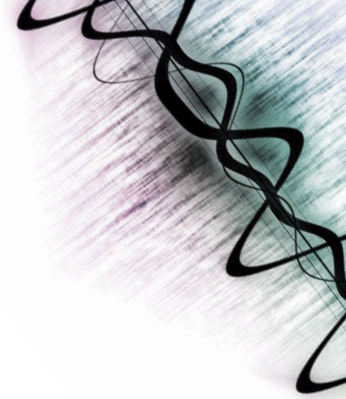


# **Tinkering Audio III**

Creative Computing– Workshop– Michael Scott

Tinkering Audio

# ECHOES



# Echoes

- Implement a simple echo and multi-echo algorithm:
  - `simple_echo(audio_data, delay)`
  - `echoes(audio_data, *delays)`

# 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.



# Adding Sounds with a Delay



```
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)
```

How can we make this a  
bit more dynamic using  
\*args ?

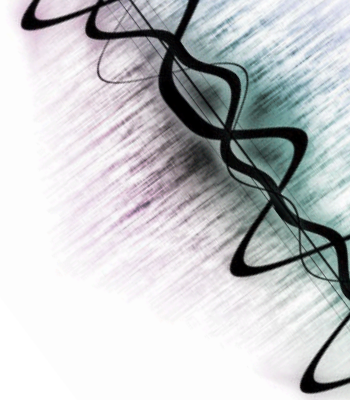
Tinkering Audio

# RESAMPLING



# Resampling

- Halve the length of the audio, in effect doubling the playback speed
- Double the length of the audio, in effect having the playback speed
- Implement a note shift



# Resampling Pseudocode

INITIALISE source index to 0

FOR EACH index IN target

    CONVERT source index to an integer

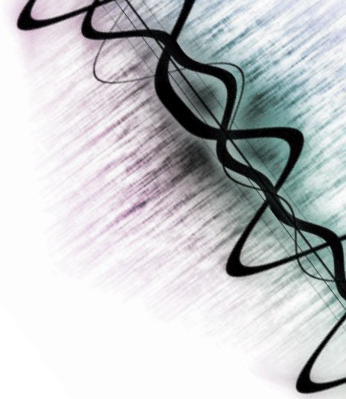
    GET the element at the relevant index from source

    PUT the element at the target index

    INCREMENT source index by a multiplier

END FOR

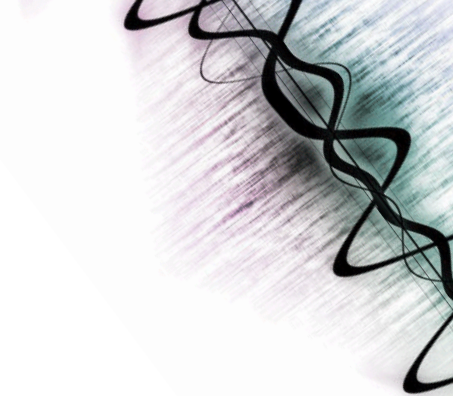
RETURN target





Tinkering Audio

# ENVELOPES



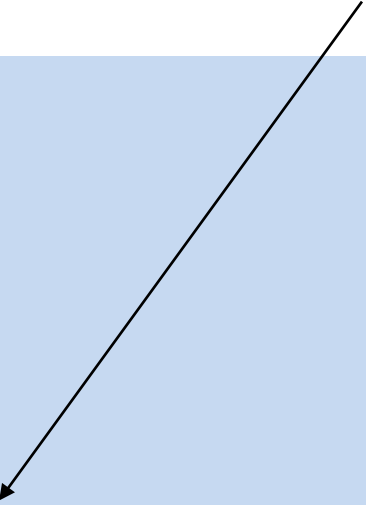
# Envelopes

- Implement `lerp()`
- Adapt your tone generation function to create an attack-decay-sustain-release envelope
  - `adsr_envelope(audio_data, attack_level, attack_time, decay_time, sustain_level, sustain_time, release_time)`

# Envelopes

What is assert?

```
def lerp(a, b, c):  
    assert 0 <= c <= 1  
    return (a * c) + (b * (1 - c))
```



# Envelopes

```
def adsr_envelope(
    audio_data,
    attack_level,
    attack_time,
    decay_time,
    sustain_level,
    sustain_time,
    release_time
):
    number_of_samples = len(audio_data)
    attack_length = int(number_of_samples * attack_time)
    decay_length = int(number_of_samples * decay_time)
    sustain_length = int(number_of_samples * sustain_time)
    release_length = int(number_of_samples * release_time)
```

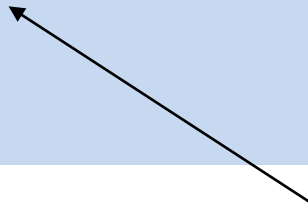
↖ You may need to adjust this to avoid gaps caused by rounding

# Envelopes

```
# Attack
p = 0
for i in range(0, attack_length):
    audio_data[i] *= lerp(0, attack_level, p /
        attack_length)
    p += 1

# Decay
p = 0
for i in range(attack_length, attack_length + decay_length):
    audio_data[i] *= lerp(attack_level, sustain_level, p /
        decay_length)
    p += 1
```

Etc...



What's this, and why?

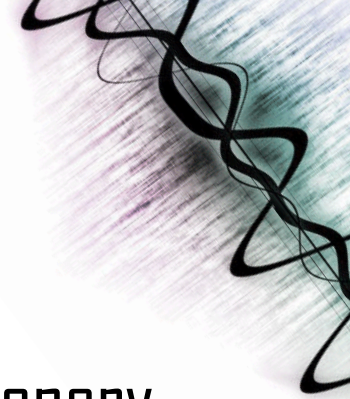
Tinkering Audio

# RANDOM MELODIES



# Random Melody

- Use the random module to select notes from a dictionary and generate a tone of that note
- Given a list of notes and note durations, we can shuffle each, play through, and repeat, to make a simple melody.
- Integrate a random number generator into the synthesizer that you have just created to randomly produce a simple melody.



# Using Choice

```
notes = {
    'A3': 220.00,
    'G3': 196.00,
    'A4': 440.00,
    'B4': 493.88,
    'C4': 523.25,
    'D4': 587.33,
    'E4': 659.25,
    'F4': 349.23,
    'G4': 392.00
}

for i in range(0,10):
    generate_tone(
        length_of_note,
        random.choice(list(notes.items()))[1]
    )
```



# Using Shuffle

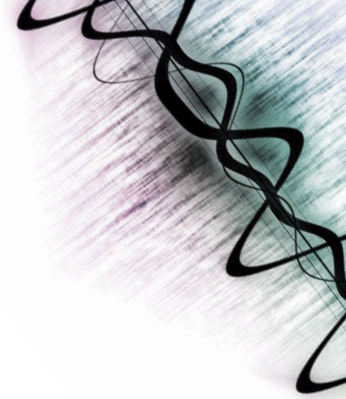
```
import random

note_list = [] # assume initialized
note_length_list = [] # also, assume initialized

for i in range(0,5):
    for j in range(0, len(note_list)):
        play(note_list[j], note_length_list[j]) # play
    random.shuffle(note_list)
    random.shuffle(note_length_list)
```

Tinkering Audio

# SYNTHESISERS



# Subtractive synthesis

- The most widely used type of synthesis in electronic music production
- A subtractive synthesizer has **oscillators** which generate tones, and **filters** which shape the frequencies – both of which can be controlled by envelopes
  - Many classic synthesizers from the 1960s/1970s were based on **analogue** oscillators and filters
  - Modern synthesizers are **digital**, although many aim to model the characteristic sound of analogue components



# A challenge (for the adventurous!)

- Incorporate subtractive synthesis (filtering) into your tinkering audio project
- Use the formulae here
  - <http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt>
  - Don't be put off by the jargon – you don't need to understand the maths behind it to use it
  - See also <http://blog.bjornroche.com/2012/08/basic-audio-egs.html>