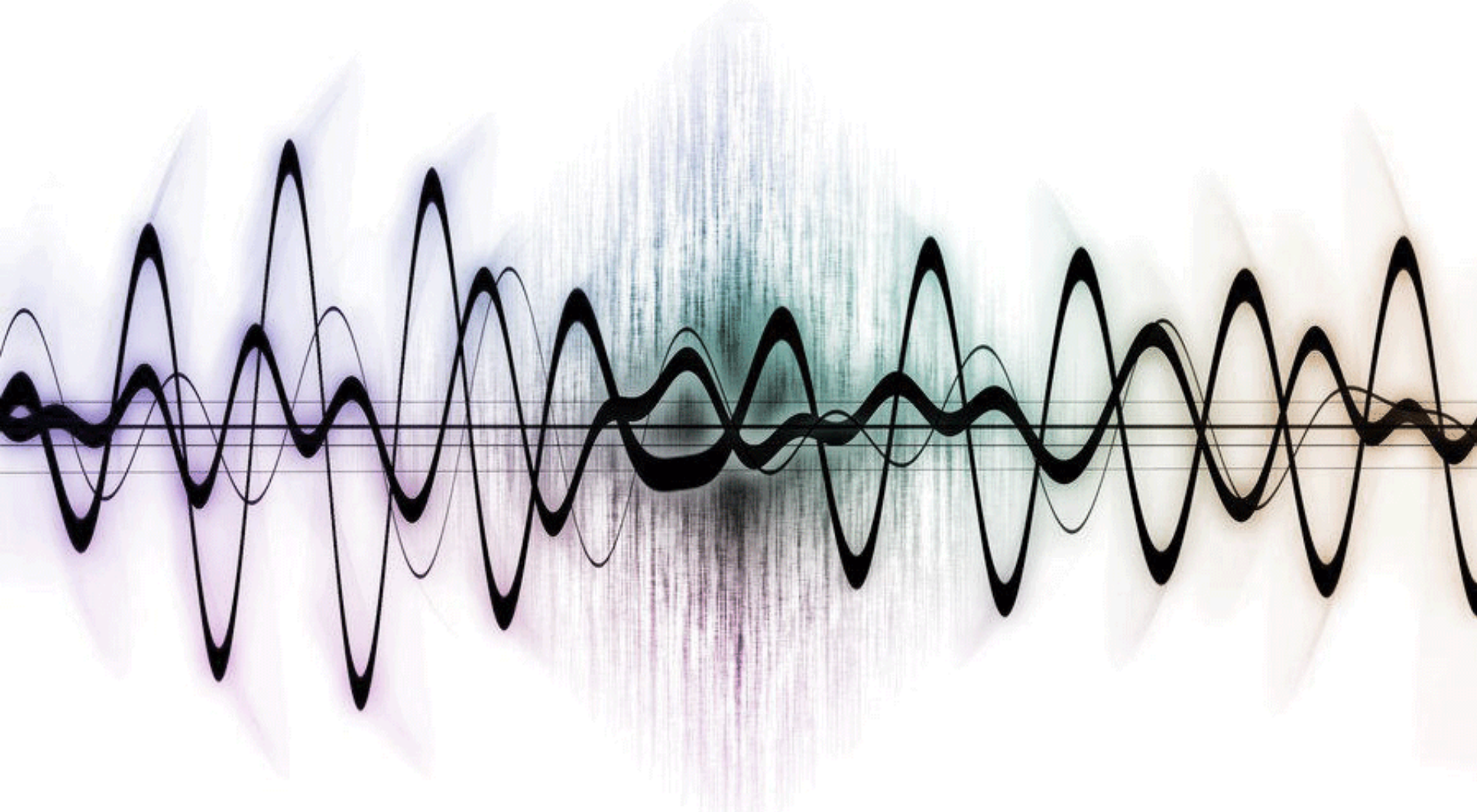


Digital Audio Workshop I

Creative Computing – Dr Michael Scott

Tone Generator Activity

- **Write** a function to develop a pure tone
- Ensure that it can generate audio at a range of 1Hz to 25,000Hz
- You will need to make use of the math module function sin() as well as functions from the wave and struct libraries

Tone Generator Activity

- Read the following documentation:

  https://docs.python.org/3.6/library/math.html
  https://docs.python.org/3.6/library/wave.html
  https://docs.python.org/3.6/library/struct.html

- Keep file formats in mind: our wav files likely use 16-bit unsigned little-endian values (i.e., `'<h'`)

- Examine wav files from the Internet using `getparams()`; and you can write your own helper functions to scan wav files to aid your investigation (e.g., max(), min(), count_sign_changes(), etc.)

Tone Generator Activity

Example

```
noise_out = wave.open(FILE, 'w')
noise_out.set_basic_paramaters()

values = []

for i in range(0, SAMPLE_LENGTH):
        value = sin(
              2.0 * PI *
              FREQUENCY *
               ( i / SAMPLE_RATE)
               ) *
               (VOLUME * BIT_DEPTH)

       packaged_value =package(FORMAT, value)

       for j in range(0, CHANNELS):
           values.append(packaged_value)

value_str = ''.join(values)
noise_out.write(value_str)

noise_out.close()
```

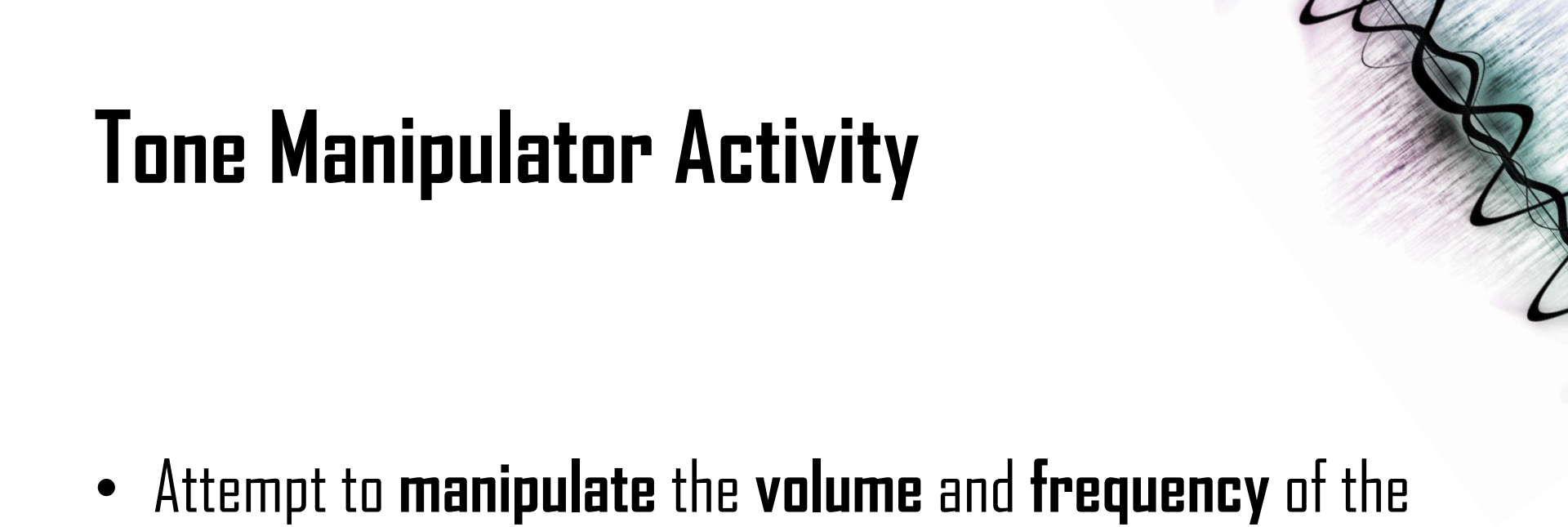
Explanation

- What should the parameters be?
- Why this formula?
  Why 2*PI?
  Why VOLUME * BIT_DEPTH?
  Why SAMPLE_RATE?
- Why append extra values?

Tone Generator Activity

- **Experiment and test** with your sounds
- Produce tones at:
  - 50Hz,
  - 150Hz,
  - 1500Hz,
  - 4000Hz
  - 8000Hz
  - 15,000Hz

Tone Manipulator Activity



- Attempt to **manipulate** the **volume** and **frequency** of the tones generated by your function.
- There is no convenient function for this in the wave module.
- Simply, manipulate the array

Tone Manipulator Activity

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

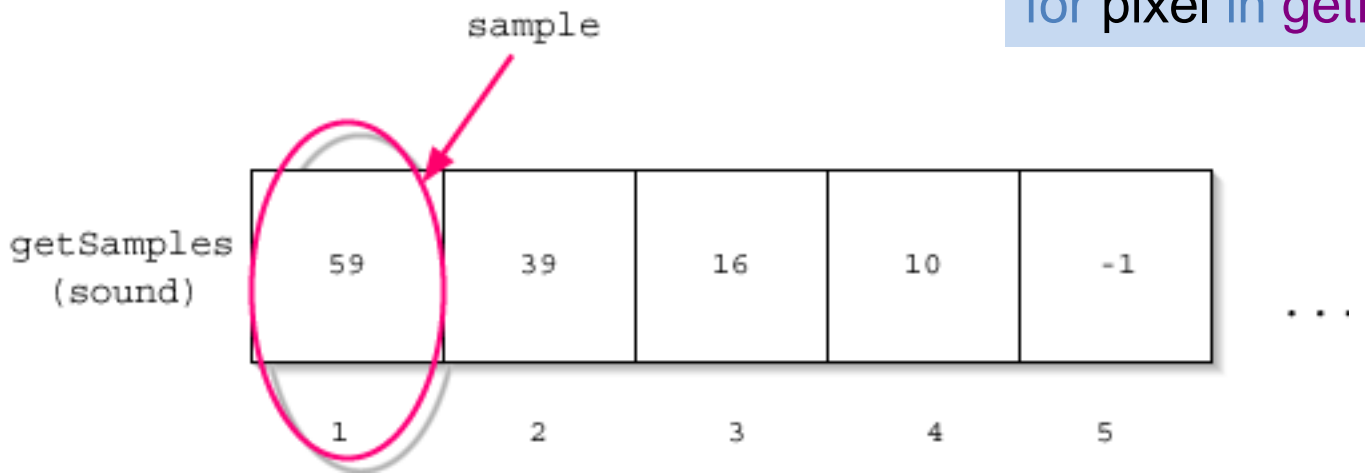
Starting the loop

- **getSamples(sound)** returns a sequence of all the sample objects in the **sound**.
- The **for** loop makes **sample** be the first sample as the block is started.

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

Compare:

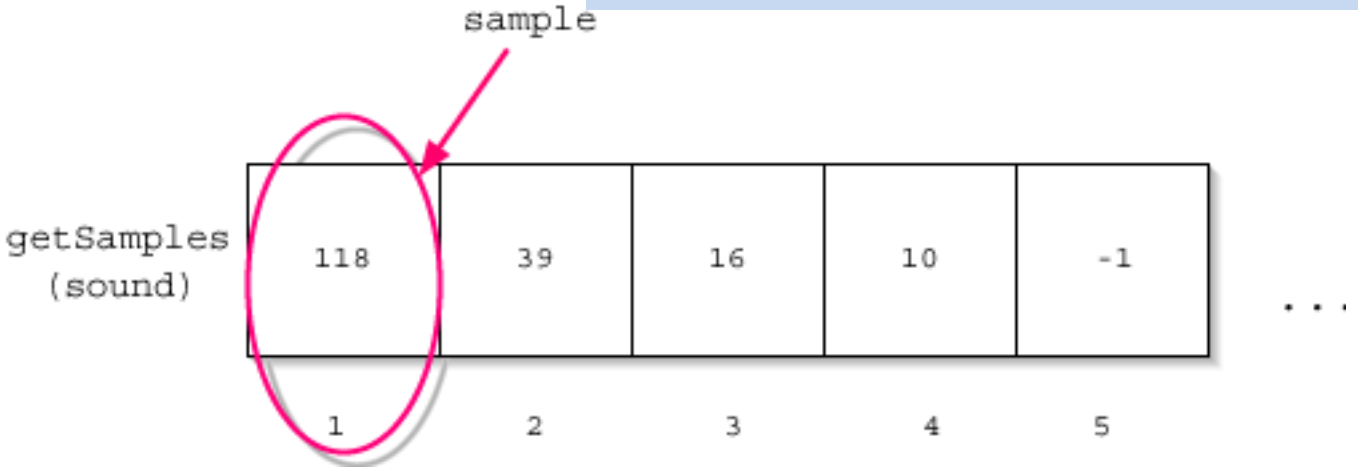
```
for pixel in getPixels(picture):
```



Executing the block

- We get the value of the sample named sample.
- We set the value of the sample to be the current value (variable value) times 2

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



Next sample

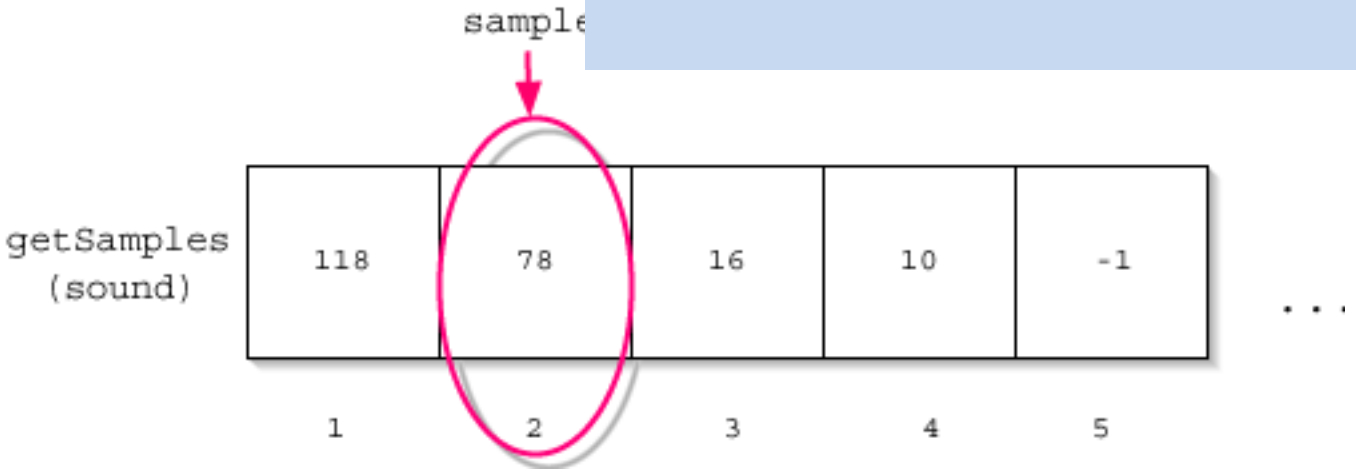
- Back to the top of the loop, and sample will now be the second sample in the sequence.

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

And increase that next sample

- We set the value of *this* sample to be the current value (variable value) times 2.

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



And on through the sequence

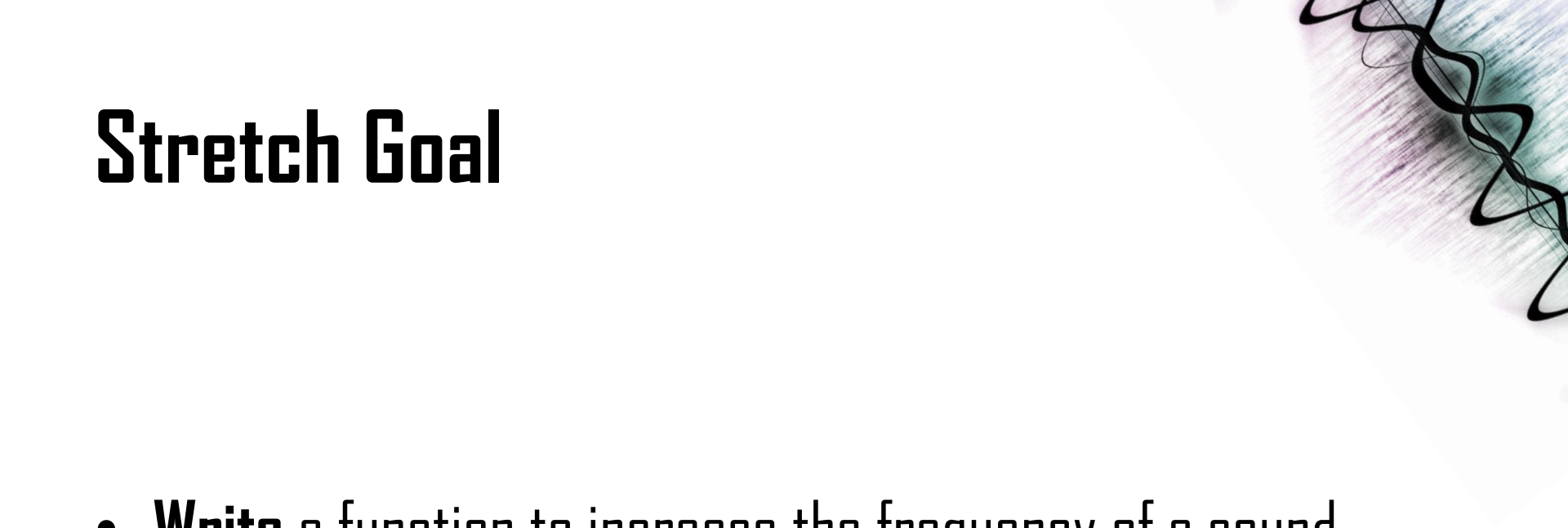
- The loop keeps repeating until *all* the samples are doubled

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

Questions

- What are the implications of a naiive multiplier?
- Is there a need for a clamp() function to ensure the minimum and maximum are not exceeded
- What is ‘clipping’?

Stretch Goal



- **Write** a function to increase the frequency of a sound
- Use the increaseVolume function as a baseline
- Hint: you will likely need to define a second, shorter array