

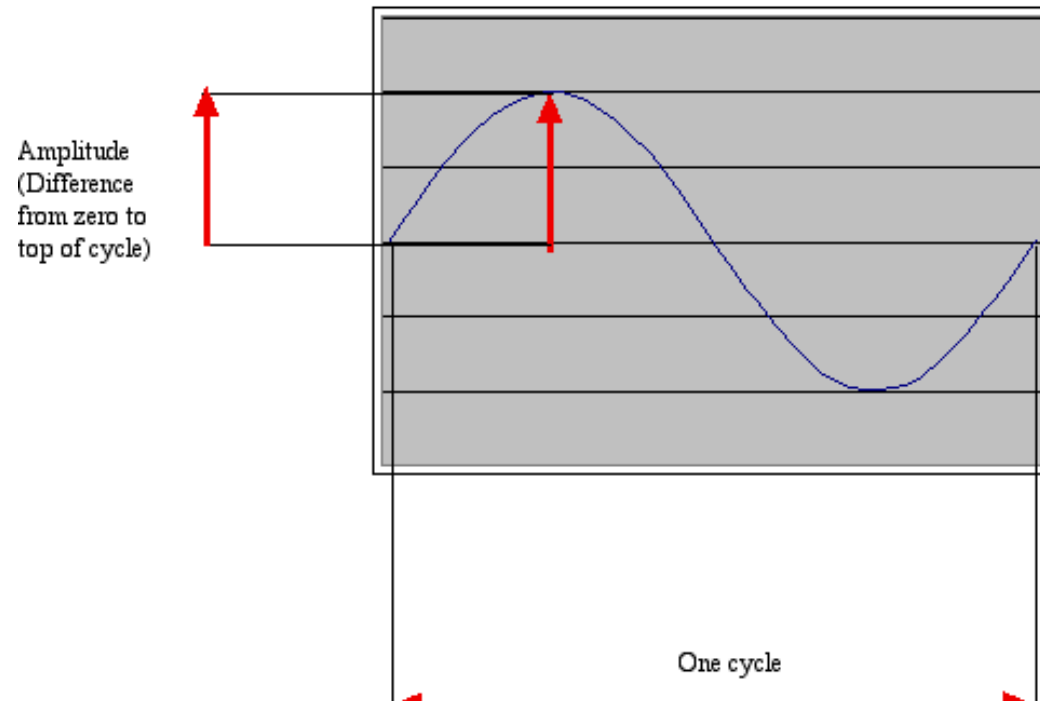


COMP2720: Automating Tools for New Media

Sound encoding and
manipulation

How sound works: Acoustics, the physics of sound

- Sounds are waves of air pressure.
 - Sound comes in cycles
 - The frequency of a wave is the number of cycles per second (cps), or *Hertz*
 - Complex sounds have more than one frequency in them.
 - The *amplitude* is the maximum height of the wave.





Volume and pitch: Psychoacoustics, the psychology of sound

- Our perception of volume is related (logarithmically) to *changes* in amplitude.
 - If the amplitude doubles, it's about a 3 decibel (dB) change.
- Our perception of *pitch* is related (logarithmically) to changes in *frequency*.
 - Higher frequencies are perceived as higher pitches.
 - We (humans) can hear between 16 Hz and 20,000 Hz (20 kHz) (some animals, eg, cows, can hear higher sounds)
 - below 4 kHz the human ear can hear the pitch difference of 1 Hz
 - The middle (first piano octave) C is 440 Hz.

“Logarithmically?”

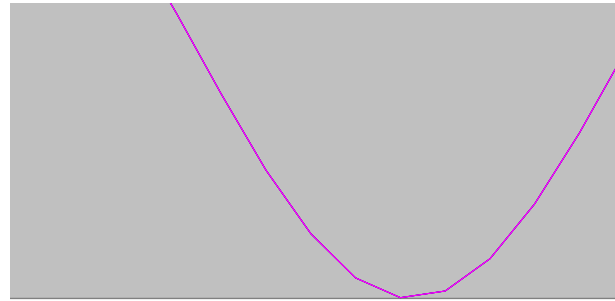
- It’s strange, but our hearing works *on ratios not differences*, e.g., for pitch.
 - We hear the difference between 200 Hz and 400 Hz as the same as the difference between 500 Hz and 1000 Hz.
 - Similarly, 200 Hz to 600 Hz, and 1000 Hz to 3000 Hz.
- Intensity (volume) is measured as *watts per meter squared*.
 - A change from 0.1W/m^2 to 0.01 W/m^2 , sounds the same to us as 0.001W/m^2 to 0.0001W/m^2 .

Decibel is a logarithmic measure

- A *decibel* is a ratio between two intensities: $10 \cdot \log_{10}(I_1/I_2)$
 - As an absolute measure, it's in comparison to threshold of audibility
 - 0 dB can't be heard.
 - Normal speech is 60 dB.
 - A shout is about 80 dB
 - An aircraft engine is about 110-120 dB

Digitising sound: How do we get that into numbers?

- Remember in calculus, estimating the curve by creating rectangles?
- We can do the same to estimate the sound curve.
 - Analog-to-digital conversion (ADC) will give us the amplitude at an instant as a number: a *sample*.
 - How many samples do we need?



Nyquist Theorem

- We need **twice as many samples as the maximum frequency** in order to represent (and recreate, later) the original sound, $N_{sr} = 2 * f_{max}$
- The number N_{sr} of samples recorded per second is the *sampling rate*.
 - If we capture 8000 samples per second, the highest frequency we can capture is 4000 Hz.
 - That's how telephones work.
 - If we capture more than 44,000 samples per second, we capture everything that we can hear (max 22,000 Hz).
 - CD quality is 44,100 samples per second.

Digitising sound in the computer

- Each sample is stored as a number (two bytes)
- What's the range of available combinations?
 - 16 bits, $2^{16} = 65,536$
 - But we want both positive and negative values.
 - To indicate compressions and rarefactions.
 - What if we use one bit to indicate positive (0) or negative (1)?
 - That leaves us with 15 bits
 - 15 bits, $2^{15} = 32,768$
 - One of those combinations will stand for zero.
 - We'll use a "positive" one, so that's one less pattern for positives.

+/- 32K

- Each sample can be between -32,768 and 32,767

Why this number? Noting bizarre:

Because $32,768 + 32,767 + 1 = 2^{16}$

< 0

> 0

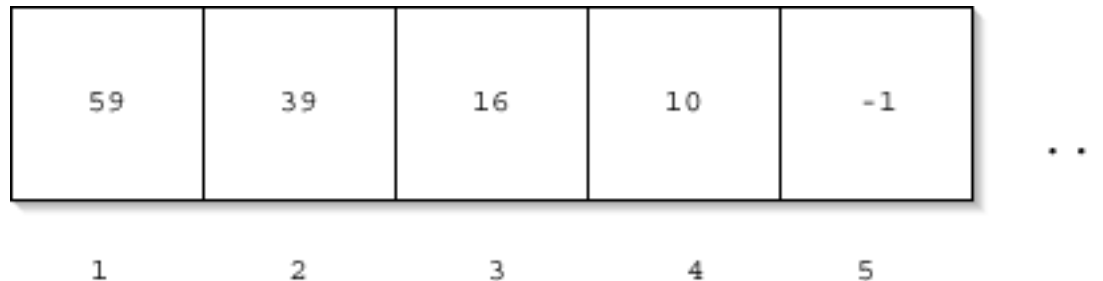
0

i.e. 16 bits, or 2 bytes

Compare this to 0..255 for light intensity
(i.e. 8 bits or 1 byte)

Sounds as arrays

- Samples are just stored one right after the other in the computer's memory.
- That's called an *array*. (Like pixels in a picture)
 - It's an especially efficient (quickly accessed) memory structure.





Working with sounds in JES

- We'll use `pickAFile` and `makeSound`.
 - We want `.wav` files (JES can also handle aiff file, but not mp3s).
- We'll use `getSamples` to get all the sample objects out of a sound.
- We can also get the value at any index with `getSampleValueAt`.
- Sounds also know their length (`getLength`) and their sampling rate (`getSamplingRate`).
- Can save sounds with `writeSoundTo(sound, "file.wav")`

Demonstrating working with sound in JES

```
>>> filename=pickAFile()
>>> print filename
/Users/guzdial/mediasources/preamble.wav
>>> sound=makeSound(filename)
>>> print sound
Sound of length 421109
>>> samples=getSamples(sound)
>>> print samples
Samples, length 421109
>>> print getSampleValueAt(sound,1)
36
>>> print getSampleValueAt(sound,2)
29
```

Demonstrating working with samples in JES

```
>>> print getLength(sound)
```

```
220568
```

```
>>> print getSamplingRate(sound)
```

```
22050.0
```

```
>>> print getSampleValueAt(sound,220568)
```

```
68
```

```
>>> print getSampleValueAt(sound,220570)
```

I wasn't able to do what you wanted.

The error `java.lang.ArrayIndexOutOfBoundsException` has occurred

Please check line 0 of

```
>>> print getSampleValueAt(sound,1)
```

```
36
```

```
>>> setSampleValueAt(sound,1,12)
```

```
>>> print getSampleValueAt(sound,1)
```

```
12
```



Working with samples

- We can get sample objects out of a sound with `getSamples(sound)` or `getSampleObjectAt(sound,index)`.
- A sample object remembers its *sound*, so if you change the sample object, the sound gets changed.
- Sample objects understand `getSample(sample)` and `setSample(sample,value)`.

Example: Manipulating samples

```
>>> soundfile=pickAFile()
>>> sound=makeSound(soundfile)
>>> sample=getSampleObjectAt(sound,1)
>>> print sample
Sample at 1 value at 59
>>> print sound
Sound of length 387573
>>> print getSound(sample)
Sound of length 387573
>>> print getSample(sample)
59
>>> setSample(sample,29)
>>> print getSample(sample)
29
```

“But there are thousands of these samples!”

- How do we do something to these samples to manipulate them, when there are thousands of them per second?
- We use a *loop* and get the computer to iterate in order to do something to each sample.
- An example loop:

```
for sample in getSamples(sound):  
    value = getSample(sample)  
    setSample(sample, value)
```

Recipe to Increase the Volume

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

Using it:

```
>>> f="/Users/guzdial/mediasources/gettysburg10.wav"  
>>> s=makeSound(f)  
>>> increaseVolume(s)  
>>> play(s)  
>>> writeSoundTo(s, "/Users/guzdial/mediasources/louder-g10.wav")
```

How did that work?

- When we evaluate `increaseVolume(s)`, the function `increaseVolume` is executed.
 - The sound in variable `s` becomes known as `sound`.
 - `sound` is a placeholder for the sound object `s`.

```
>>> f=pickAFile()
>>> s=makeSound(f)
>>> increaseVolume(s)
```

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

Compare:

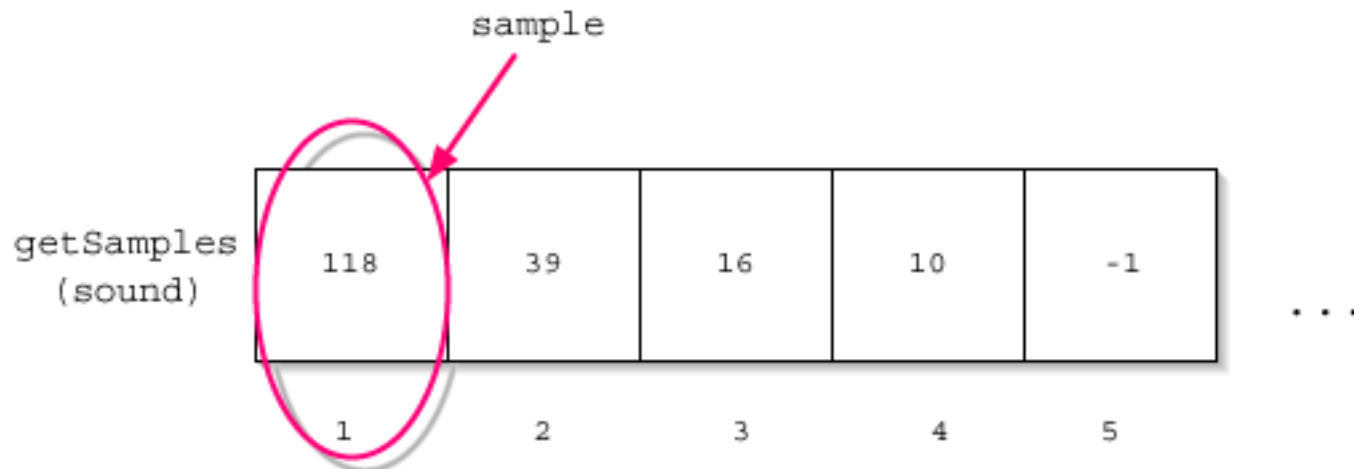
```
for pixel in getPixels(picture):
```



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.

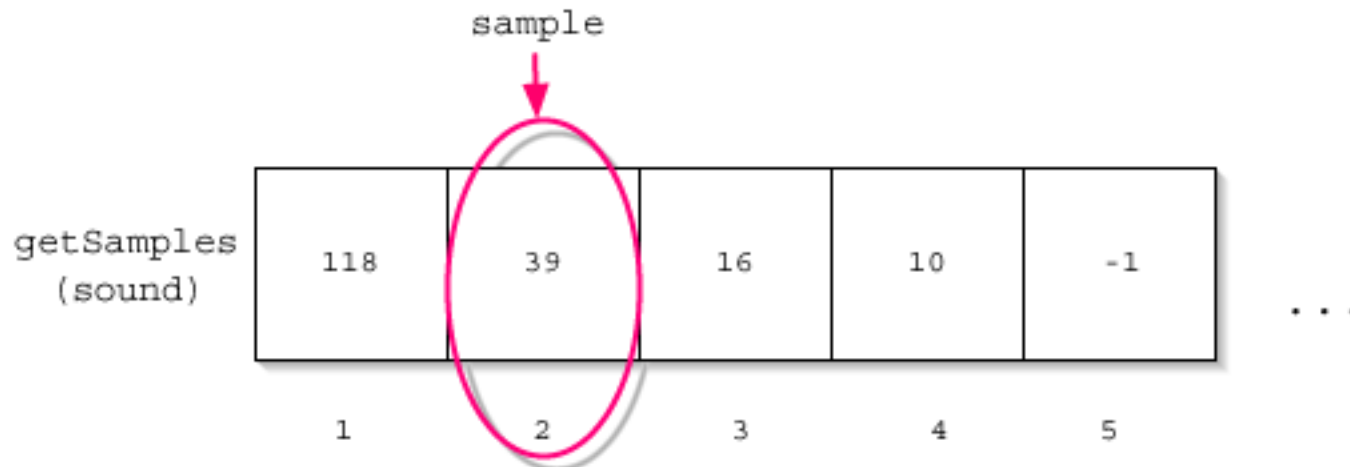
```
def increaseVolume(sound):  
    for sample in getSamples(sound):  
        value = getSample(sample)  
        setSample(sample, value * 2)
```



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 = getSample(sample)  
        setSample(sample, value * 2)
```



And increase that next sample

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

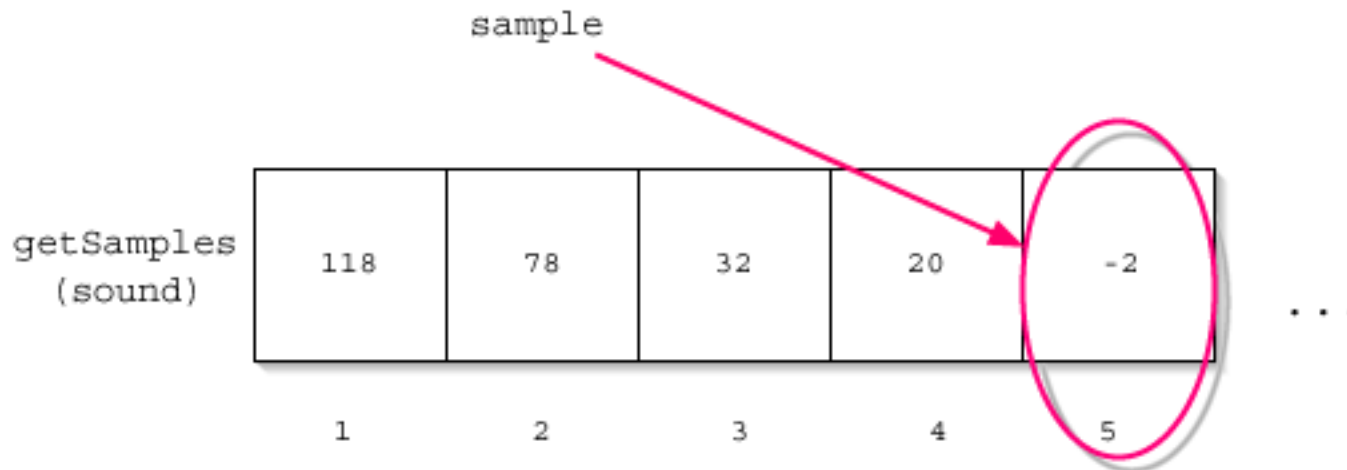
```
def increaseVolume(sound):  
    for sample in getSamples(sound):  
        value = getSample(sample)  
        setSample(sample, value * 2)
```



And on through the sequence

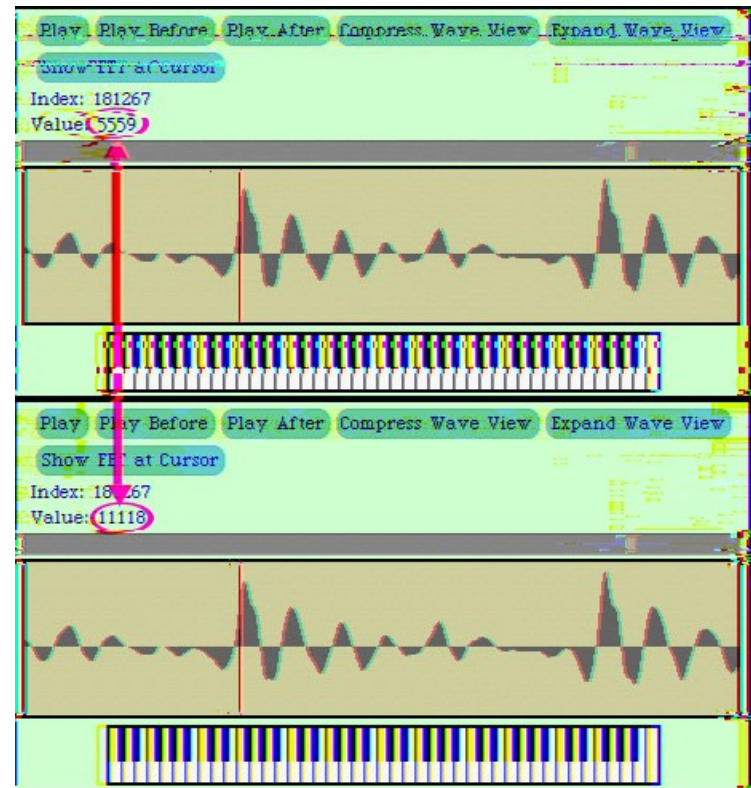
- The loop keeps repeating until *all* the samples are doubled.

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



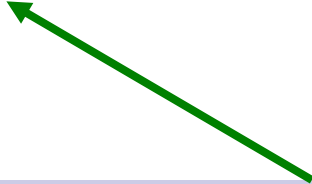
How are we *sure* that that worked?

```
>>> print s
Sound of length 220567
>>> print f
/Users/guzdial/mediasources/gettysburg10.wav
>>> soriginal=makeSound(f)
>>> print getSampleValueAt(s,1)
118
>>> print getSampleValueAt(soriginal,1)
59
>>> print getSampleValueAt(s,2)
78
>>> print getSampleValueAt(soriginal,2)
39
>>> print getSampleValueAt(s,1000)
-80
>>> print getSampleValueAt(soriginal,1000)
-40
```



Decreasing the volume

```
def decreaseVolume(sound):  
    for sample in getSamples(sound):  
        value = getSample(sample)  
        setSample(sample, value * 0.5)
```



This works *just* like increaseVolume, but we're *lowering* each sample by 50% instead of doubling it.

Recognise some similarities?

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



```
def increaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p,value*1.2)
```

```
def decreaseVolume(sound):  
    for sample in getSamples(sound):  
        value = getSample(sample)  
        setSample(sample,value * 0.5)
```



```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p,value*0.5)
```

Does increasing the volume change the *volume* setting?

- No!

- The physical volume setting indicates an upper bound, the potential loudest sound.
- Within that potential, sounds can be louder or softer.
 - They can fill that space, but might not.

(Have you ever noticed how commercials are always louder than regular programs?)

- Louder content attracts your attention.
- It maximizes the *potential* sound.