



COMP2720: Automating Tools for New Media

Using `range` to manipulate
samples

Maximising volume

- How do we get maximal volume?
 - For example, automatic recording level.
- It's a three-step process:
 - First, figure out the loudest sound (largest sample).
 - Next, figure out how much we have to increase/decrease that sound to fill the available 'space'.
 - We want to find the amplification factor amp , where $\text{amp} * \text{loudest} = 32767$
 - In other words: $\text{amp} = 32767/\text{loudest}$
 - Finally, amplify each sample by multiplying it by amp

Maxmising (normalising) the sound

```
def normalise(sound):  
    largest = 0  
    for s in getSamples(sound):  
        largest = max(largest, getSample(s) )  
    amplification = 32767.0 / largest  
  
    print "Largest sample value in original sound was", largest  
    print "Amplification multiplier is", amplification  
  
    for s in getSamples(sound):  
        louder = amplification * getSample(s)  
        setSample(s, louder)
```

This loop finds the loudest sample

Q: Why 32767.0?
A: Later...

This loop actually amplifies the sound

Max()

- `max()` is a function that takes any number of inputs, and always returns the largest.
- There is also a function `min()` which works similarly but returns the minimum.

```
>>> print max(1,2,3)
```

```
3
```


```
>>> print max(4,67,98,-1,2)
```

```
98
```

Or: use if instead of max

```
def normalise(sound):  
    largest = 0  
    for s in getSamples(sound):  
        if getSample(s) > largest:  
            largest = getSample(s)  
    amplification = 32767.0 / largest
```

Instead of finding max of all samples, check each in turn to see if it's the largest so far



```
print "Largest sample value in original sound was", largest  
print "Amplification factor is", amplification
```

```
for s in getSamples(sound):  
    louder = amplification * getSample(s)  
    setSample(s, louder)
```



Aside: Positive and negative extremes assumed to be equal

- We're making an assumption here that the maximum positive value is also the maximum negative value.
 - That should be true for the sounds we deal with, but isn't necessarily true
- Try adding a constant to every sample.
 - That makes it non-cyclic.
 - I.e. the compressions and rarefactions in the sound wave are not equal.
 - But it's fairly subtle what's happening to the sound.

Why 32767.0, not 32767?

- Why do we divide out of 32767.0 and not just simply 32767?
 - Because of the way Python handles numbers.
 - If you give it integers, it will only ever compute integers.

```
>>> print 1.0/2  
0.5
```

```
>>> print 1.0/2.0  
0.5
```

```
>>> print 1/2  
0
```



Avoiding clipping

- Why are we being so careful to stay within range? What if we just multiplied all the samples by some big number and let some of them go over 32,767?
- The result then is *clipping*
 - Clipping: The awful, buzzing noise whenever the sound volume is beyond the maximum that your sound system can handle.



Knowing where we are in the sound

- More complex operations require us to know where we are in the sound, which *sample*.
 - Not just process all the samples exactly the same.
- Examples:
 - Reversing a sound.
 - It's just copying, like we did with pixels.
 - Changing the frequency of a sound.
 - Using *sampling*, like we did with pixels.
 - Splicing sounds.

Processing only *parts* of the sound

- What if we wanted to increase or decrease the volume of only part of the sound?
- Q: How would we do it?
- A: We'd have to use a `range()` function with our for loop.
 - Just like when we manipulated only parts of a picture by using `range()` in conjunction with `getPixels()`

Using for to count with range()

```
>>> print range(1,3)
[1, 2]
>>> print range(3,1)
[]
>>> print range(-1,5)
[-1, 0, 1, 2, 3, 4]
>>> print range(1,100)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... 99]
```

Increasing volume by sample index

```
def increaseVolumeByRange(sound):  
    for sampleNumber in range(1, getLength(sound)+1):  
        value = getSampleValueAt(sound, sampleNumber)  
        setSampleValueAt(sound, sampleNumber, value * 2)
```

This really is the same as:

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

Recipe to play a sound backwards (Trace it!)

```
def playBackward(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)
```

Start at end
of sound



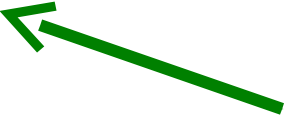
```
    srcSample = getLength(source)  
    for destSample in range(1, getLength(dest)+1):  
        srcVolume = getSampleValueAt(source, srcSample)  
        setSampleValueAt(dest, destSample, srcVolume)  
        srcSample = srcSample - 1
```

Work backward



```
    return dest
```

Return the processed sound for further use
in the function that calls playBackward



How does this work?

- We make two copies of the sound,
- The `srcSample` starts at the end, and the `destSample` goes from 1 to the end.
- Each time through the loop, we copy the sample value from the `srcSample` to the `destSample`.

Note that the `destSample` is *increasing* by 1 each time through the loop, but `srcSample` is *decreasing* by 1 each time through the loop!

```
def playBackward(filename):
    source = makeSound(filename)
    dest = makeSound(filename)

    srcSample = getLength(source)
    for destSample in range(1, getLength(dest)+1):
        srcVolume = getSampleValueAt(source, srcSample)
        setSampleValueAt(dest, destSample, srcVolume)
        srcSample = srcSample - 1

    return dest
```

Starting out (3 samples here)

```
def playBackward(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename) ← You are here  
  
    srcSample = getLength(source)  
    for destSample in range(1, getLength(dest)+1):  
        srcVolume = getSampleValueAt(source, srcSample)  
        setSampleValueAt(dest, destSample, srcVolume)  
        srcSample = srcSample - 1  
  
    return dest
```

12	25	13
----	----	----

source

12	25	13
----	----	----

dest

Ready for the copy

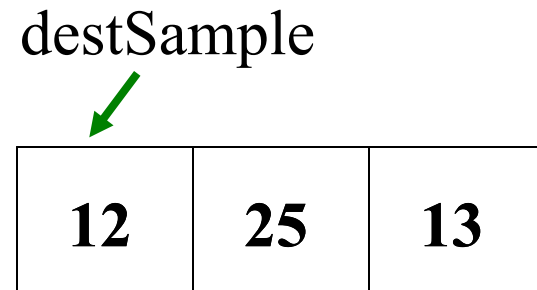
```
def playBackward(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)  
  
    srcSample = getLength(source)  
    for destSample in range(1, getLength(dest)+1):  
        srcVolume = getSampleValueAt(source, srcSample)  
        setSampleValueAt(dest, destSample, srcVolume)  
        srcSample = srcSample - 1  
  
    return dest
```



You are here



source



dest

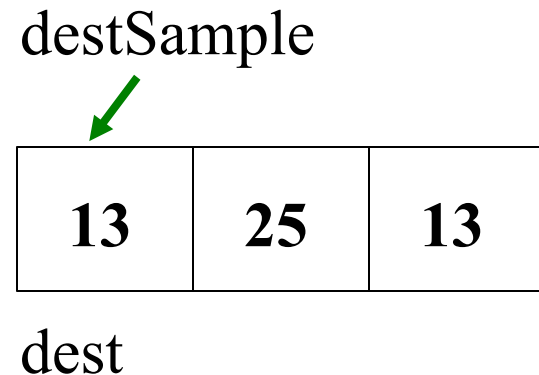
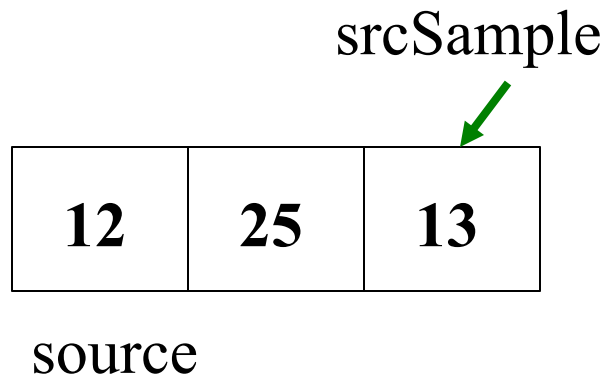
Do the copy

```
def playBackward(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)
```

```
    srcSample = getLength(source)  
    for destSample in range(1, getLength(dest)+1):  
        srcVolume = getSampleValueAt(source, srcSample)  
        setSampleValueAt(dest, destSample, srcVolume)  
        srcSample = srcSample - 1
```

← You are here

```
    return dest
```

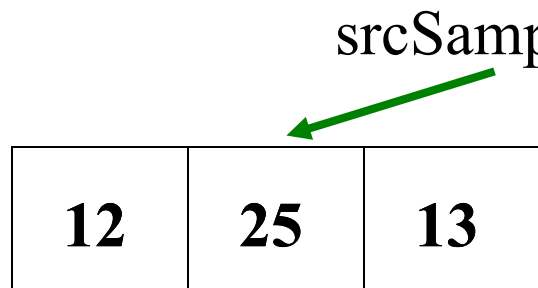


Ready for the next one?

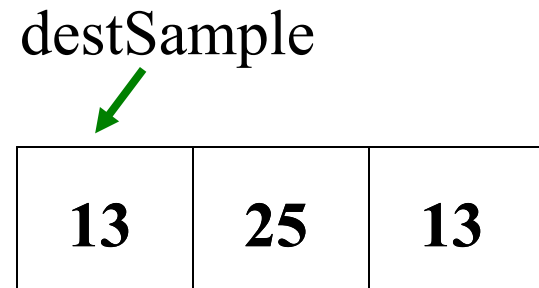
```
def playBackward(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)  
  
    srcSample = getLength(source)  
    for destSample in range(1, getLength(dest)+1):  
        srcVolume = getSampleValueAt(source, srcSample)  
        setSampleValueAt(dest, destSample, srcVolume)  
        srcSample = srcSample - 1  
  
    return dest
```



You are here



source



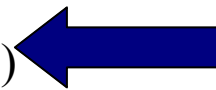
dest

Moving them together

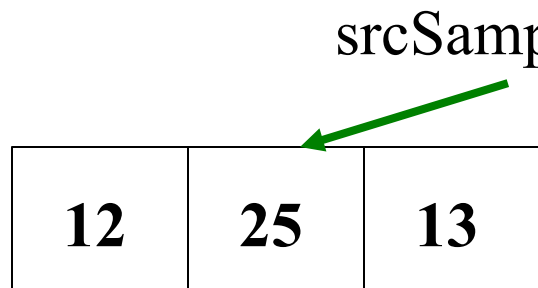
```
def playBackward(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)
```

```
    srcSample = getLength(source)  
    for destSample in range(1, getLength(dest)+1):  
        srcVolume = getSampleValueAt(source, srcSample)  
        setSampleValueAt(dest, destSample, srcVolume)  
        srcSample = srcSample - 1
```

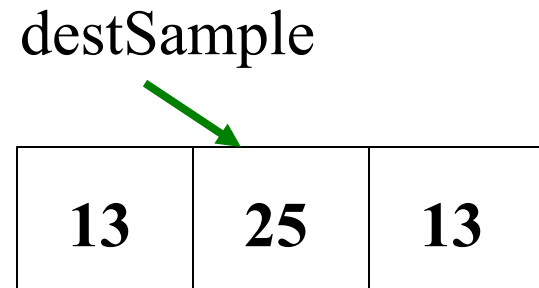
```
    return dest
```



You are here



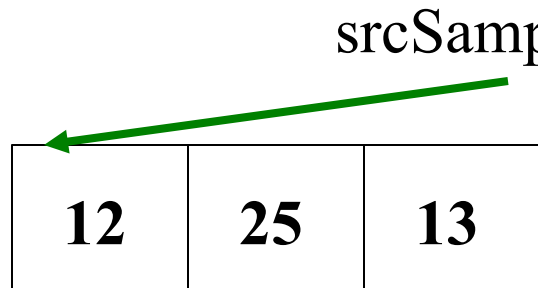
source



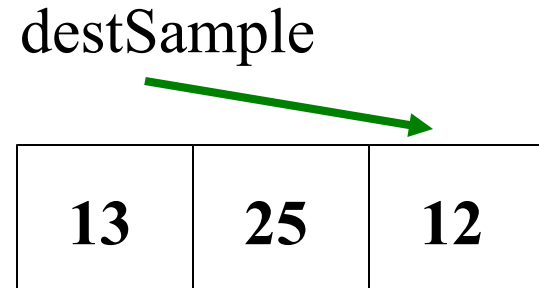
dest

How we end up

```
def playBackward(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)  
  
    srcSample = getLength(source)  
    for destSample in range(1, getLength(dest)+1):  
        srcVolume = getSampleValueAt(source, srcSample)  
        setSampleValueAt(dest, destSample, srcVolume)  
        srcSample = srcSample - 1  
  
    return dest
```



source



dest

Recipe for halving the frequency of a sound


```
def half(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)
```

This is how a
sampling synthesizer
works!

```
    srcSample = 1  
    for destSample in range(1, getLength(dest)+1):  
        volume = getSampleValueAt(source, int(srcSample) )  
        setSampleValueAt(dest, destSample, volume)  
        srcSample = srcSample + 0.5
```

```
    play(dest)  
    return dest
```

Here are
the piece
that do it



Changing pitch of sound vs. changing picture size

1

```
def half(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)
```

```
    srcSample = 1  
    for destSample in range(1, getLength(dest)+1):  
        vol = getSampleValueAt( source, int(srcSample))  
        setSampleValueAt(dest, destSample, vol)  
        srcSample = srcSample + 0.5
```

```
    play(dest)  
    return dest
```

3

```
def copyBarbsFaceLarger():  
    barbf=getMediaPath("barbara.jpg")  
    barb = makePicture(barbf)  
    canvas = makeEmptyPicture(400,600)  
    sourceX = 45  
    for targetX in range(100,100+((200-45)*2)):  
        sourceY = 25  
        for targetY in range(100,100+((200-25)*2)):  
            px = getPixel(barb,int(sourceX),int(sourceY))  
            color = getColor(px)  
            setColor(getPixel(canvas,targetX,targetY), color)  
            sourceY = sourceY + 0.5  
            sourceX = sourceX + 0.5  
        show(barb)  
    show(canvas)  
    return canvas
```

1

2

3




Both of them are sampling

- Both of them have three parts:
 - A start where objects are set up.
 - A loop where samples or pixels are copied from one place to another.
 - To decrease the frequency or the size, we take each sample/pixel twice.
 - In both cases, we do that by incrementing the index by 0.5 and taking the integer of the index.
 - Finishing up and returning the result.

Recipe to double the frequency of a sound

Here's the critical piece:
We skip every other
sample in the source!

```
def double(filename):
    source = makeSound(filename)
    target = makeSound(filename)
    targetIndex = 1
    for sourceIndex in range(1, getLength(source)+1, 2):
        setSampleValueAt( target, targetIndex,
                          getSampleValueAt( source, sourceIndex))
        targetIndex = targetIndex + 1
    # Clear out the rest of the target sound -- it's only half full!
    for secondHalf in range( getLength( target)/2, getLength( target)):
        setSampleValueAt(target,targetIndex,0)
        targetIndex = targetIndex + 1
    play(target)
    return target
```



What happens if we don't "clear out" the end?

Try this out!

```
def double(filename):
    source = makeSound(filename)
    target = makeSound(filename)
    targetIndex = 1
    for sourceIndex in range(1, getLength(source)+1, 2):
        setSampleValueAt( target, targetIndex,
                          getSampleValueAt( source, sourceIndex))
        targetIndex = targetIndex + 1
    #Clear out the rest of the target sound -- it's only half full!
    #for secondHalf in range( getLength( target)/2, getLength( target)):
    # setSampleValueAt(target,targetIndex,0)
    # targetIndex = targetIndex + 1
    play(target)
    return target
```

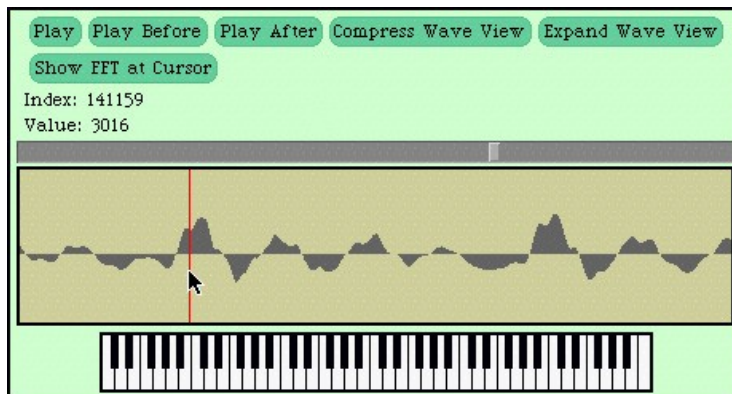
"Switch off" these lines of code by commenting them out.

Splicing Sounds

- *Splicing* gets its name from literally cutting and pasting pieces of magnetic tape together.
- Doing it digitally is easy (in principle), but painstaking.
- Say we want to splice pieces of speech together:
 - We find where the end points of words are.
 - We copy the samples into the right places to make the words come out as we want them.
 - (We can also change the volume of the words as we move them, to increase or decrease emphasis and make it sound more natural.)

Finding the word end-points

- Using **MediaTools** and play before/after cursor, can figure out the index numbers where each word ends.



Word	Ending index
We	15730
the	17407
People	26726
of	32131
the	33413
United	40052
States	55510

Now, it's all about copying

- We have to keep track of the source and target indices, `srcSample` and `destSample`

```
destSample = Where-the-incoming-sound-should-start  
for srcSample in range(startingPoint, endingPoint):  
    sampleValue = getSampleValueAt(source, srcSample)  
    setSampleValueAt(dest, destSample, sampleValue)  
    destSample = destSample + 1
```

The whole splice

```
def splicePreamble():
    file = "/Users/guzdial/mediasources/preamble10.wav"
    source = makeSound(file)
    dest = makeSound(file) # This will be the newly spliced sound
    destSample=17408      # targetIndex starts at just after "We the" in the new sound
    for srcSample in range( 33414, 40052): # Where the word "United" is in the sound
        setSampleValueAt(dest, destSample, getSampleValueAt( source, srcSample))
        destSample = destSample + 1
    for srcSample in range(17408, 26726): # Where the word "People" is in the sound
        setSampleValueAt(dest, destSample, getSampleValueAt( source, srcSample))
        destSample = destSample + 1
    for index in range(1, 1000):          #Stick some quiet space after that
        setSampleValueAt(dest, destSample, 0)
        destSample = destSample + 1
    play(dest)                            #Let's hear and return the result
    return dest
```

What's going on here?

- First, set up a source and target.
- Next, we copy “United” (samples 33,414 to 40,052) after “We the” (sample 17,408)
 - That means that we end up at $17,408 + (40,052 - 33,414) = 17,408 + 6,638 = 24,046$
 - Where does “People” start?
- Next, we copy “People” (17,408 to 26,726) immediately afterward.
 - Do we have to copy “of” to?
 - Or is there a pause in there that we can make use of?
- Finally, we insert a little (1/441-th of a second) of space – 0's

Word	Ending index
We	15730
the	17407
People	26726
of	32131
the	33413
United	40052
States	55510

What if we didn't do that second copy? Or the pause?

```
def splicePreamble():
    file = "/Users/guzdial/mediasources/preamble10.wav"
    source = makeSound(file)
    dest = makeSound(file) # This will be the newly spliced sound
    destSample=17408      # targetIndex starts at just after "We the" in the new sound
    for srcSample in range( 33414, 40052): # Where the word "United" is in the sound
        setSampleValueAt(dest, destSample, getSampleValueAt( source, srcSample))
        destSample = destSample + 1
    #for srcSample in range(17408, 26726): # Where the word "People" is in the sound
    #setSampleValueAt(dest, destSample, getSampleValueAt( source, srcSample))
    #destSample = destSample + 1
    #for index in range(1, 1000):          #Stick some quiet space after that
        #setSampleValueAt(dest, destSample, 0)
        #destSample = destSample + 1
    play(dest)                            #Let's hear and return the result
    return dest
```



Changing the splice

- What if we wanted to increase or decrease the volume of an inserted word?
 - Simple! Multiply each sample by something as it's pulled from the source.
- Could we do something like slowly increase volume (emphasis) or normalise the sound?
 - Sure! Just like we've done in past programs, but instead of working across all samples, we work across only the samples in that sound!

Making more complex sounds

- We know that natural sounds are often the combination of multiple sounds.
- Adding waves in physics or math is hard.
- In computer science, it's easy! Simply add the samples at the same index in the two waves:

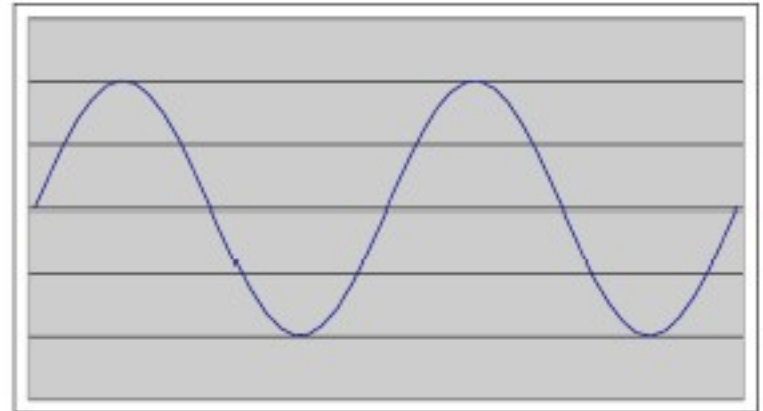
```
for srcSample in range(1, getLength(source)+1):  
    destValue=getSampleValueAt(dest, srcSample)  
    srcValue=getSampleValueAt(source,srcSample)  
    setSampleValueAt(source, srcSample, srcValue+destValue)
```

Adding sounds

The first two are sine waves generated in Excel.

The third is just the sum of the first two columns.

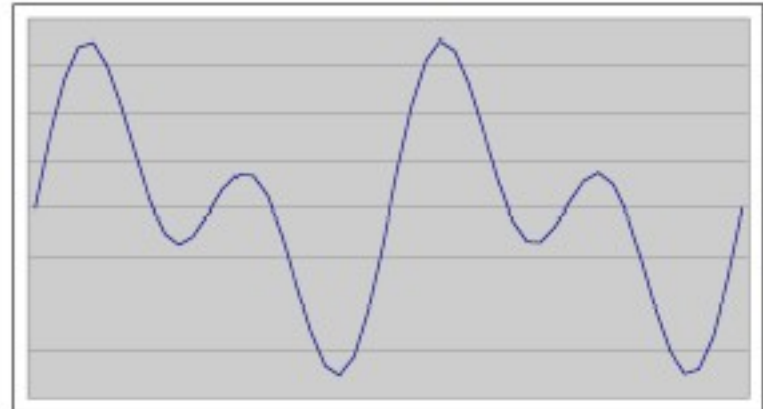
a



b



$$\mathbf{a + b = c}$$





Uses for adding sounds

- We can mix sounds
 - We even know how to change the volumes of the two sounds, even over time (e.g., fading in or fading out)
- We can create echoes.
- We can add sine (or other) waves together to create kinds of instruments/sounds that do not physically exist, but which sound interesting and complex.

A function for adding two sounds

```
def addSoundInto(sound1, sound2):
```

```
    for sampleNmr in range(1, getLength(sound1)+1):  
        sample1 = getSampleValueAt(sound1, sampleNmr)  
        sample2 = getSampleValueAt(sound2, sampleNmr)  
        setSampleValueAt(sound2, sampleNmr, sample1 + sample2)
```

Notice that this adds sound1 and sound2
by adding sound1 *into* sound2

Making a chord by mixing three notes

```
>>> setMediaPath()
```

```
New media folder: C:\Documents and Settings\Mark Guzdial\My  
Documents\mediasources\
```

```
>>> getMediaPath("bassoon-c4.wav")
```

```
'C:\\Documents and Settings\\Mark Guzdial\\My  
Documents\\mediasources\\bassoon-c4.wav'
```

```
>>> c4=makeSound(getMediaPath("bassoon-c4.wav"))
```

```
>>> e4=makeSound(getMediaPath("bassoon-e4.wav"))
```

```
>>> g4=makeSound(getMediaPath("bassoon-g4.wav"))
```

```
>>> addSoundInto(e4,c4)
```

```
>>> play(c4)
```

```
>>> addSoundInto(g4,c4)
```


```
>>> play(c4)
```

Adding sounds with a delay

```
def makeChord(sound1, sound2, sound3):  
    for index in range(1, getLength(sound1)):  
        s1Sample = getSampleValueAt(sound1, index)  
        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)
```

- Add in sound2 after 1000 samples
- Add in sound3 after 2000 samples

Note that in this version we are adding into sound1!



What to do now

- Read chapters 6, 7 and 8 on sounds.
- Start working on home work 1.
- Start working on assignment 1 (seriously)!