INFT1004 - SEMESTER 1 - 2017		LECTURE TOPICS	
Week 1	Feb 27	Introduction, Assignment, Arithmetic	
Week 2	Mar 6	Sequence, Quick Start, Programming Style	
Week 3	Mar 13	Pictures, Functions, Media Paths	
Week 4	Mar 20	Arrays, Pixels, For Loop, Reference Passing	
Week 5	Mar 27	Nested Loops, Selection, Advanced Pictures	
Week 6	Apr 3	Lists, Strings, Input & Output, Files	Practical Test
Week 7	Apr 10	Drawing Pictures, Program Design, While Loop	Assignment set
Recess	Apr 14 – Apr 23	Mid Semester Recess Break	
Week 8	Apr 24	No Lecture / Revision and Assignment in Labs	
Week 9	May 1	Data Structures, Processing sound	
Week 10	May 8	Advanced sound	Assignment part 1 due 8:00am Tue, May 9
Week 11	May 15	Movies, Scope, Import	
Week 12	May 22	Turtles, Writing Classes	Assignment part 2 due 8:00am Tue, May 23
Week 13	May 29	Revision	
Mid Year Examination Period - MUST be available normal & supplementary period			

Lecture Topics and Lab topics are the same for each week

INFT1004

Visual Programming

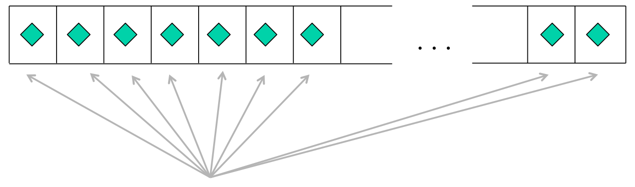
Module 10.1
Advanced Sound

Guzdial & Ericson - Third Edition - chapters 7 and 8)
Guzdial & Ericson - Fourth (Global) Edition – chapters 8 and 9)

Revision
Sound and Samples

Just as a digitised picture is a grid of individual pixels that look like a continuous image.

A digitised sound is a collection of individual samples that sound like a continuous sound.



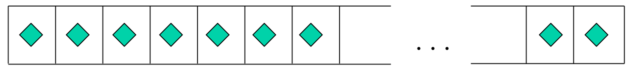
samples

Revision
Sound and Samples

Just as a digitised picture is a grid of individual pixels that look like a continuous image.

A digitised sound is a collection of individual samples that sound like a continuous sound.

samples

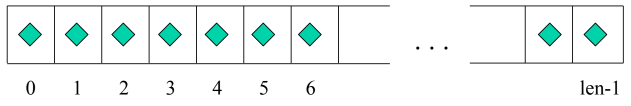


As with pixels, there is generally a huge number of them – 20,000 or more samples for every second of sound. (44,000 for CD quality)

Revision
Sound and Samples

JES lets us store the samples in an array, where we can adjust, for example, the amplitude (and thus the volume) of some or all of the samples

```
samples = getSamples(mySound)
```



Now let's do some more with sounds . . .

Example Sound Code

In week 9 and week 10 lectures there is a fair amount of code related to sound processing.

This week's lecture will give an overview of some of the key bits for this code.

Remember the tutorials for week 9, 10 are also helpful in understanding how to work with sound in JES.

(Even though you are busy with assignments – sound will be in exam – don't ignore this!)

Module 10.1 Code

```
###  Make sure you set the mediaPath before using
###  spliceSentence() and spliceSentence2()
###  Its expects to find the wav file "NowGoodMen.wav"
###  in this path
###  spliceSentence2() also creates new wav files in this path

       spliceSentence()
       spliceSentence2()

###  Note some of these functions are used by other functions
       normalise()
       clip()
       copy()
```

Mod10_01_AdvancedSound.py

Module 10.1 Code

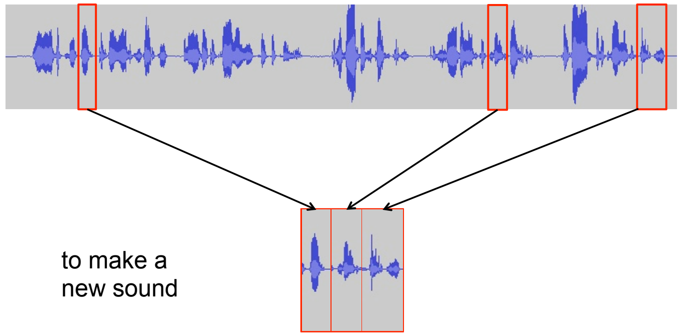
```
###  Some more fancy sound processing - check
###  definitions for parameters and return values
###  You should be able to follow this code now
###  read it, understand it and use it

   reverse()
   mirror()
   mix()
   echo()
   frequencyShift()
```

Mod10_01_AdvancedSound.py

Splicing Sounds

The sound lesson here is to copy bits from one or more existing sounds



to make a new sound

Splicing Sounds

The book gives one example; we'll use another: making a different message from a recording of...

```
"Now is the time for all good men to come to
the aid of the party."
```

NowGoodMen.wav

↓

Mod10_01_AdvancedSound.py - NowGoodMen.wav

Splicing Sounds

The book gives one example; we'll use another: making a different message from a recording of...

```
"Now is the time for all good men to come to
the aid of the party."
```

↓

```
"time to party."
```

We copy bits from highly specific ranges in the original sound

How do we know which indexes?

Splicing Sounds

We explore the original sound, play various selections, and look at the start and end of each selection we're interested in.

start end

```
"Now is the time for all good men to come to
the aid of the party."
```

↓

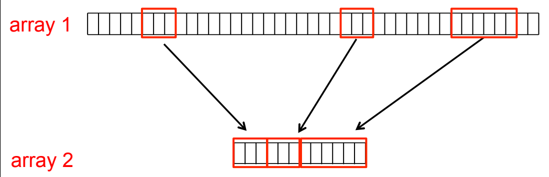
```
"time to party."
```

We copy bits from highly specific ranges in the original sound

How do we know which indexes?

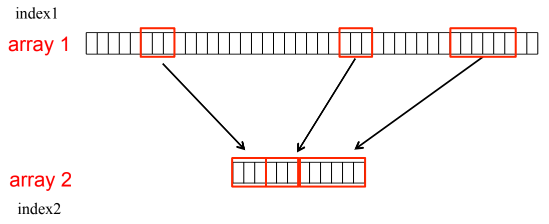
Copying Arrays

To do this we need to copy some samples from one array to another array.



Copying Arrays

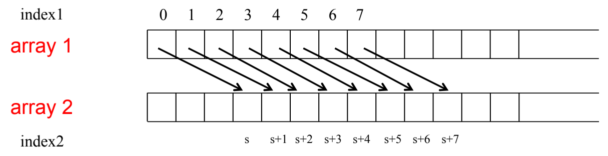
To do this we need to copy some samples from one array to another array.



Note that we have to keep track of two different index positions – one in each array

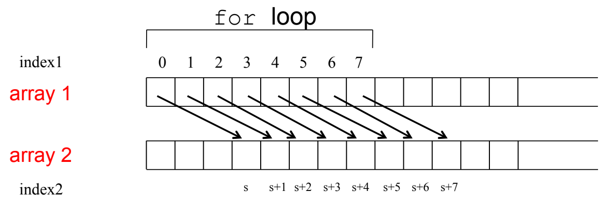
Copying Arrays

Indeed when we copy from one array to another we typically need to keep track of different positions (indexes) in each array



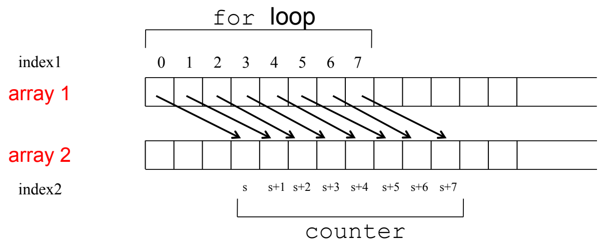
e.g. here we are copying from the beginning of one array into the middle of another one

Copying Arrays



We really only want to go through the for loop once – we just want to keep track of two different index positions

Copying Arrays



So for the second array we have separate index that keeps count of where it is up to in the loop; we start it at starting position we need ; and simply add 1 to it after each sample

How well does it work?

The quality of the spliced sound is seldom very good.

For example, we run words together, so it’s hard to separate them.

For example, we use different intonation in different parts of a sentence.

We even pronounce the same word differently compare “the” in “the aid” and “the party”

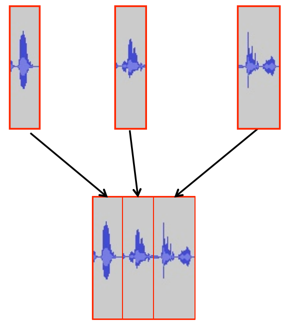
How well does it work?

Clever splicing is a real art!

But what you’re learning about here is arrays, indexes, and copying!

Clipping bits & saving them

A different approach to the splicing would be to have each word in its own file.



Clipping bits & saving them

`clip()` makes a new sound by copying a specified range from an existing sound

It still needs exactly the same range start and finish numbers as `splice()`

But combining the clips is now a little easier

See `spliceSentence2()`

Mod10_01_AdvancedSound.py

Clipping bits & saving them

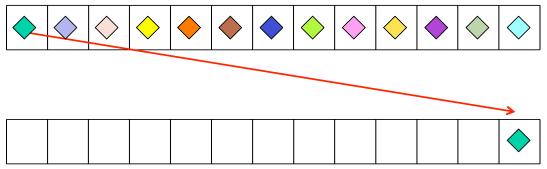
This approach is probably no better for this specific task

But the general function to clip and save, and the one to copy one sound to another, are more generally useful

So we will probably find a lot more chances to reuse these functions for other tasks

Reversing a sound

As we continue to explore the use of arrays and indexes, reversal is a nice exercise

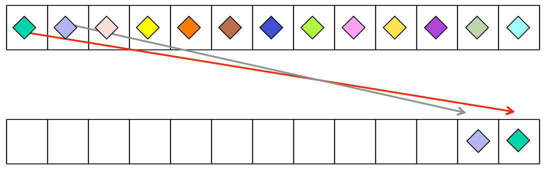


We want the first element of the original to be the last of the copy, etc

Mod10_01_AdvancedSound.py

Reversing a sound

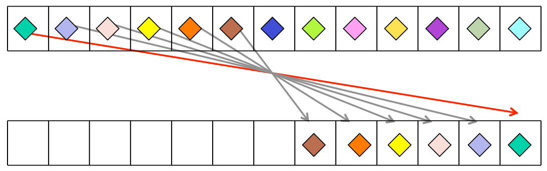
As we continue to explore the use of arrays and indexes, reversal is a nice exercise



We want the first element of the original to be the last of the copy, etc

Reversing a sound

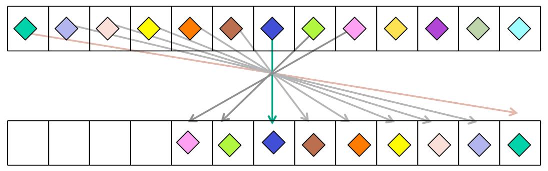
As we continue to explore the use of arrays and indexes, reversal is a nice exercise



We want the first element of the original to be the last of the copy, etc

Reversing a sound

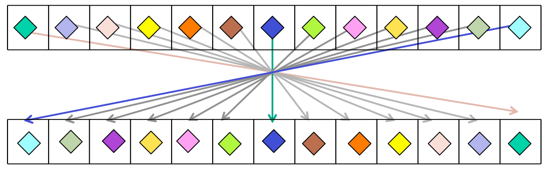
As we continue to explore the use of arrays and indexes, reversal is a nice exercise



We want the first element of the original to be the last of the copy, etc

Reversing a sound

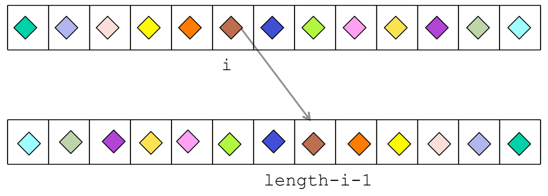
As we continue to explore the use of arrays and indexes, reversal is a nice exercise



We want the first element of the original to be the last of the copy, etc

Reversing a sound

A little problem-solving tells us that index `i` in the original gets copied to index `length-i-1` in the copy; (or we could maintain two separate indexes)



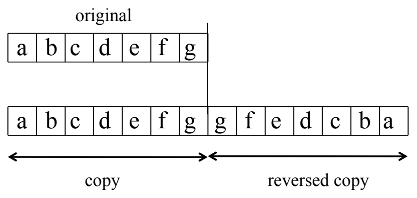
Try function reverse() – it makes some interesting sounds

Mod10_01_AdvancedSound.py

Mirror a sound

Mirroring a sound is exactly the same principle as mirroring a picture.

Leave one half as it is, and make the other half a reversed copy of it.



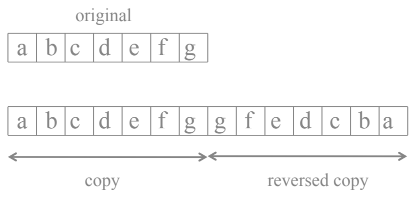
Mirror a sound

To leave one half as it is, we just copy the samples.

To make the other half a mirror of the first, we copy

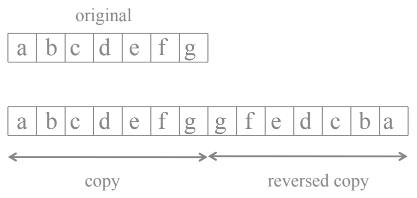
from index 0 to index length – 1
from index 1 to index length – 2
....

from index i to index length – 1 – i (in general)



Mirror a sound

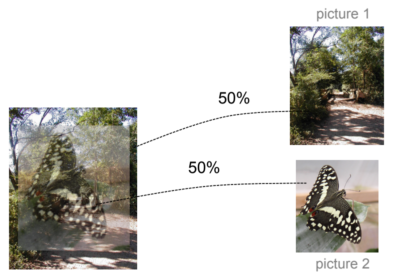
While the textbook changes the sound itself, I've come to prefer working with and returning a copy of the sound – which leaves the original as it was. (No side effects)



Mixing sounds

Remember how we mixed pictures?

50% of this pixel's R, G, and B and 50% of that pixel's R, G, and B mixes the two so that it looks like one transparent picture laid over another

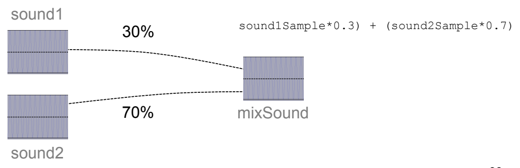


Mixing sounds

It works exactly the same with sound:

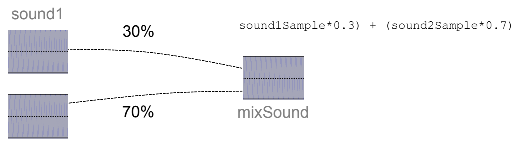
50% of the sample from one sound and 50% of the sample form another is a 50/50 mix of the two sounds

Different proportions (eg 30/70) emphasise one sound over the other



Mixing sounds

See mix().



Mod10_01_AdvancedSound.py

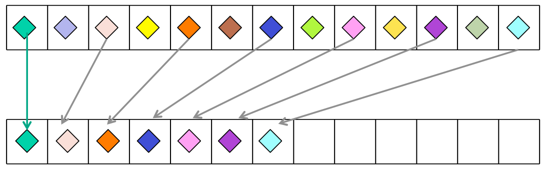
How well does it work?

You can sometimes hear odd things when using this mix function - but it's not bad for a start.

Remember, this is not a processing tool for professionals; it's an illustration of arrays, applying them to sound samples so that we can see/hear a real effect

Shifting frequency

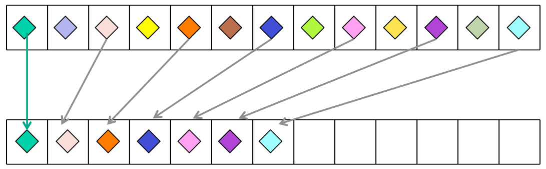
Frequency shifting is trickier than amplitude adjustment
Instead of adjusting the values of individual samples, we have to reduce or increase the number of samples



Mod10_01_AdvancedSound.py

Higher frequency

To reduce the number of samples (increase frequency), we have to skip some of the samples in the original file

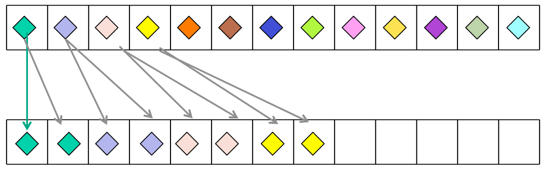


That's all right – that's what sampling is all about: taking little snapshots of the sound at specific moments

The new sound will clearly be shorter

Reducing frequency

To increase the number of samples, each sample from the original sound has to be copied more than once into the new sound



The new sound will clearly be longer (lower in frequency)

Shifting frequency

The textbook has a very nice way of shifting the frequency of any sound by a specified factor

As with all the programs in the book (and the lecture demo programs), if you make the effort to understand it, you'll learn a lot

But be careful – when modifying these they need to be tested to make sure they work with the requirements!