Media Computing with Python

Byung Kim and Fred Martin
Computer Science Dept.
UMass, Lowell
{kim, fredm}@cs.uml.edu

Media Computing with Python

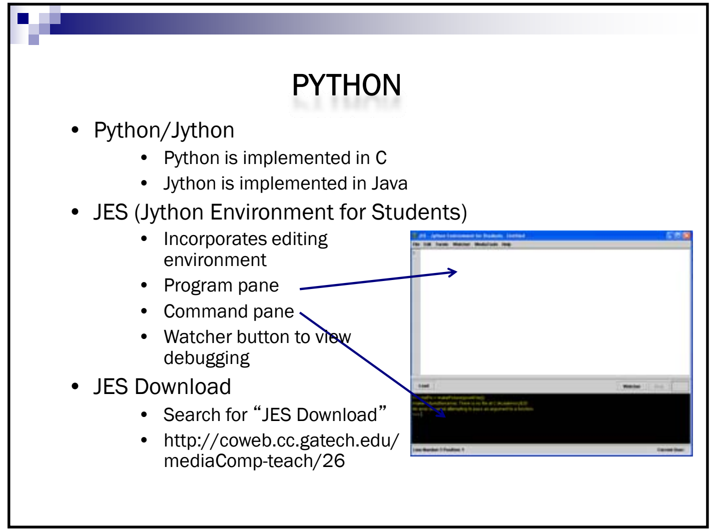
- Computer Programming Can Be Fun
  - Mark Guzdial, Georgia Tech
- Problem space -- Digital media
  - Text, image, video, audio
- Computer Programming
  - Process (steps) of manipulating data
  - Python language

Table of Contents

- Introduction to Python with JES (Jython Enviroment for Students)
- Python functions
- Colors and pictures
  - Change colors in pictures
  - Make a negative of a picture
- Sounds
  - Reverse an audio file

PYTHON

- Python/Jython
  - Python is implemented in C
  - Jython is implemented in Java
- JES (Jython Environment for Students)
  - Incorporates editing environment
  - Program pane
  - Command pane
  - Watcher button to view debugging
- JES Download
  - Search for "JES Download"
  - http://coweb.cc.gatech.edu/mediaComp-teach/26



Simple Computation in JES (+*/-%)

- Addition
  3 + 4
- Multiplication
  3 * 4
- Division
  3 / 4
- Subtraction
  3 – 4
- Negation
  -4
- Modulo (Remainder)
  10 % 2 and 11 % 2

Why is 3/2 = 1 ?

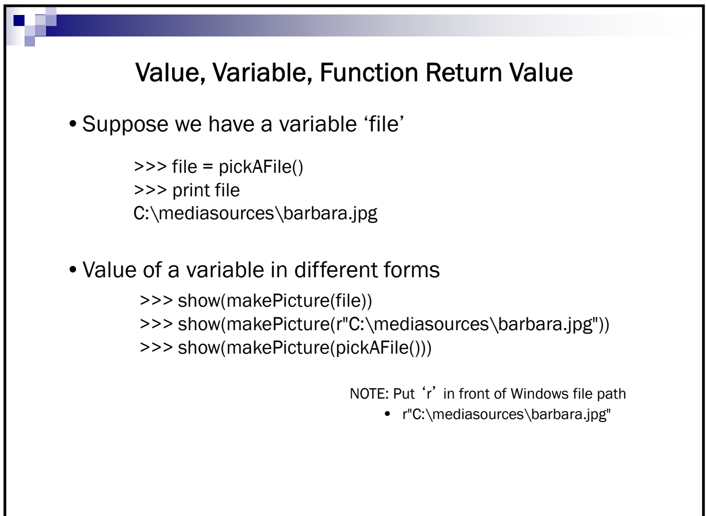
- Python is a *typed* language
  - Each value has a type associated with it
  - Tells the computer how to interpret the number
- Built-in Types
  - int: 235, -2, 39402, etc.
  - float: 3.25, -52.95, etc.
  - string:
    - >>> "Hello World!"
    - >>> 'Hi, there !'
    - >>> "don't do that"
    - >>> 'I said "Stop!"'
    - >>> "Hey" + " you!"
  - boolean: true or falst

Variables

- Name a value and re-use it
  - >>> x = 2*8
  - >>> x/2
  - >>> x
  - Whenever a variable name appears, it is substituted by its value
  - A variable name can be reused, redefined
- Naming convention
  - MUST start with a letter followed by alphanumeric
    - aVar, cs100, …
  - CASE matters
    - 'Print' is not the same as 'print'
    - 'makePicture' is not the same as 'makepicture', nor as 'Makepicture'
  - Multi-word name
    - First letter is capitalized
    - Convention, not absolute rule
  - Use sensible, meaningful name

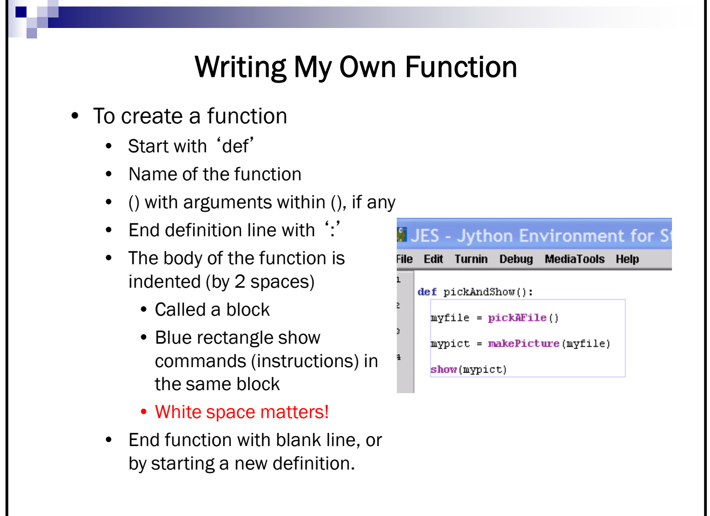
JES Functions

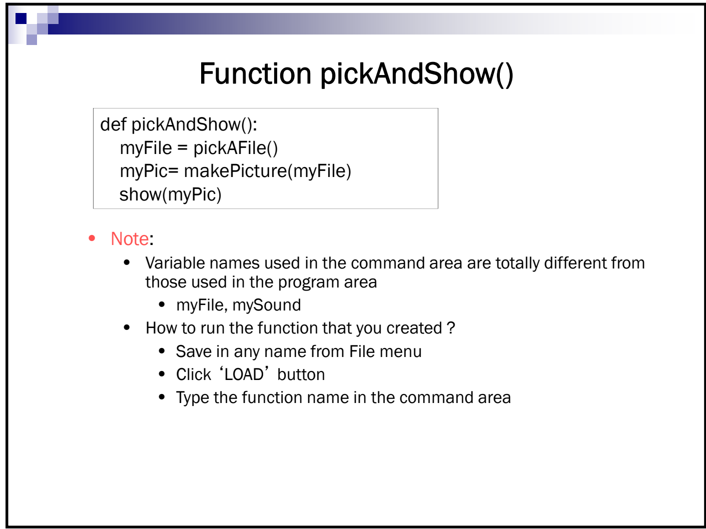
- Functions pre-defined in JES for image and sound manipulation
  - pickAFile(): allows a user to select a file
  - makePicture(): creates and returns a picture object
  - show(): display a picture in the argument
  - makeSound(): creates and returns an audio object
  - play(): plays out a sound
- A function returns a value
  - Like a quadratic $f(x) = x^2+5x-10$, a sine(90), ..
- A variable vs. a function
  - Variable can be reused with the same value (until re-assigned)
  - Function can be reused to return different values according to arguments (some functions do not have arguments)

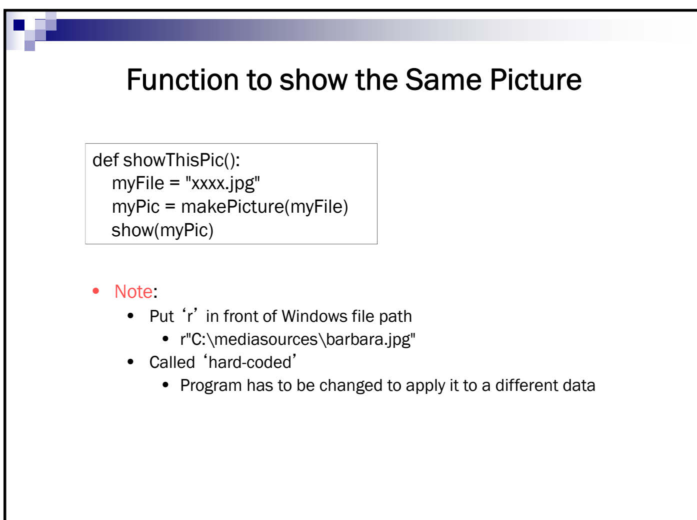


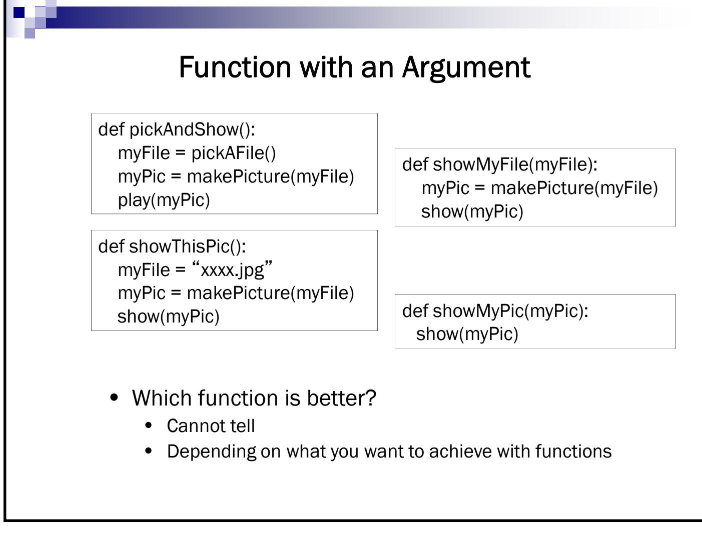
LAB 1

- In JES Command area,
  - Select any '.jpg' file from C:\mediasources
  - Convert the selected file into a picture file using 'makePicture()'
  - Display the picture using 'show()'
  - Select any '.wav' file from C:\mediasources
  - Convert the selected file into a sound file using 'makeSound()'
  - Replay the sound using 'play()'



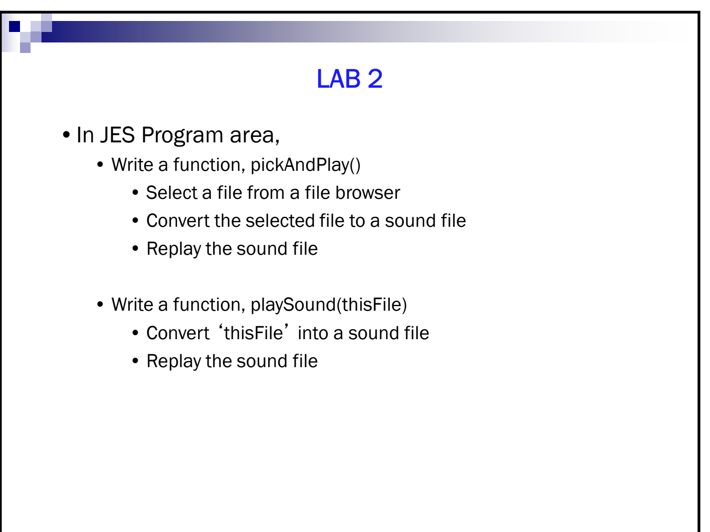


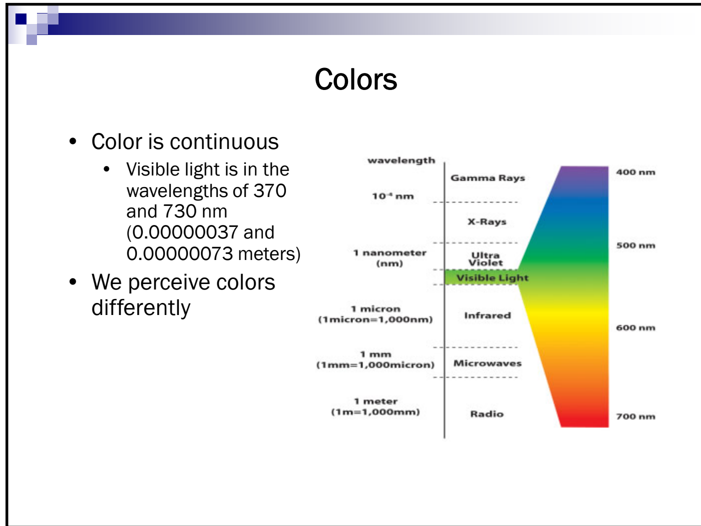


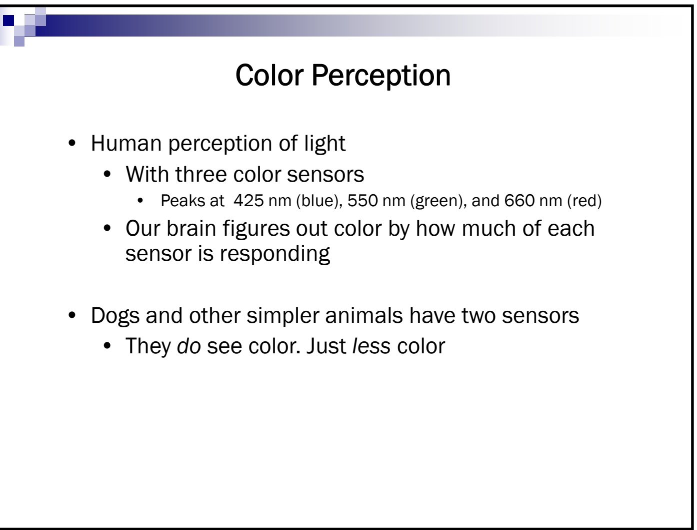


What Can Go Wrong ?

- Are names correct ?
  - Spelling and CASE
- Indentation
  - Instructions in the same block have to be indented by the same amount
- Variable Names
  - Variables in the command and the program areas are NOT related







Luminance vs. Color

- Luminance is a measure of light intensity
  - Luminance allows us to perceive borders of things, motion, depth
  - Luminance perception is *color blind*.
- Brightness is our perception of luminance
  - brightness is *not* the amount of light, but our *perception* of the amount of light.
  - We see blue as "darker" than red, even if same amount of light.
  - Much of our luminance perception is based on comparison to backgrounds, not raw values.
    - White's illusion
      - Same gray luminance
      - Appears brighter in black stripe
- Different parts of the brain perceive color and luminance.

Digital Pictures

- Digitized as a bunch of dots (squares)
  - With enough dots, it looks continuous
  - Our eyes have limited resolution
  - Our background/depth *acuity* is particulary low
- Each picture element is referred to as a *pixel*

Pixels

- Pixels are *picture elements*
  - Each pixel object "knows" its *color*
    - e.g. given a pixel, a Python function can get the color out of it.
  - It also "knows" where it is in its *picture*
    - e.g. given a pixel and a picture, a Python function can find out where the pixel is located in the picture
- A picture is a *matrix* of pixels
  - With rows of pixels as *arrays*
  - With two dimensions: Width and Height
  - We need a two-dimensional array: a *matrix*

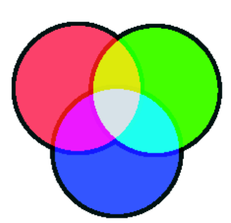
Referencing a Matrix

	1	2	3	4
1	15	12	13	10
2	9	7		
3	6			

- We talk about positions in a matrix as (x,y), or (horizontal, vertical)
- Element (2,1) in the matrix at left is the value 12
- Element (1,3) is 6

Color Encoding

- Each pixel encodes color at that position in the picture
- Most common for computers
  - RGB (Red, Green, Blue)
  - Each does appear as a separate dot on most devices, but our eye blends them.
  - In most computer-based models of RGB, a single *byte* (8 bits) is used for each
    - Total RGB color is 24 bits



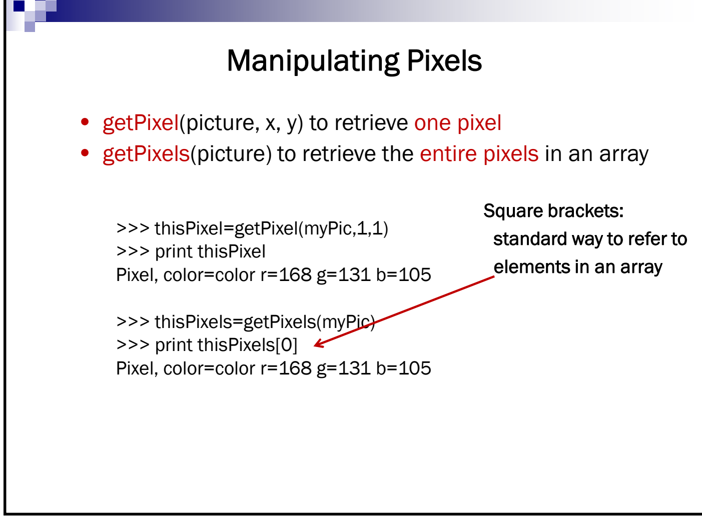
Encoding RGB

- Colors go from (0,0,0) to (255,255,255)
  - If all components have the same values -> grayscale
    - (50,50,50) at (2,2)
  - (0,0,0) at (1,2) is black
  - (255,255,255) is white
- Colors are represented in 24 bit
  - That is 16,777,216 (2^{24}) possible colors
  - Our eyes can discern millions of colors -> close
  - But, we don't get 16 million colors from computer monitors

	1	2	3
1	100,10,5	5,10,100	255,0,0
2	0,0,0	50,50,50	0,100,0

What is a Picture in JES ?

- A picture object in JES
  - Is an encoding that represents a picture
  - Knows its height and width
    - Knows how many pixels it has in both directions
  - Knows its file name
    - A picture is not a file
    - Only when you call makePicture(), it becomes a picture
    - But it knows which file it came from
  - Knows its window when opened by show() or repaint()



What Can We Do with a Pixel

- getRed, getGreen, and getBlue are functions that
  - return a color value (between 0 and 255) at a specified pixel
- setRed, setGreen, and setBlue are functions that
  - set its color value at a specified pixel
- We can also get, set, and make colors
  - getColor returns a Color object with three color values at a pixel
  - setColor sets the pixel to the specified color
  - makeColor returns a Color object with specified three color values
  - pickAColor lets you use a color chooser and returns the chosen color
- We also have functions that can makeLighter and makeDarker an input color

Example

```
>>> thisPixel = getPixel(myPic,1,1)
>>> print thisPixel
Pixel, color=color r=168 g=131 b=105

# get/set individual color values
>>> print getRed(thisPixel)
168
>>> setRed(thisPixel,255)
>>> print getRed(thisPixel)
255
>>> color=getColor(thisPixel)
>>> print color
color r=255 g=131 b=105
>>> setColor(thisPixel,color)
```

```
>>> newColor=makeColor(0,100,0)
>>> print newColor
color r=0 g=100 b=0
>>> setColor(thisPixel,newColor)
>>> print getColor(thisPixel)
color r=0 g=100 b=0
>>> print color
color r=168 g=131 b=105
>>> print makeDarker(color)
color r=117 g=91 b=73
>>> print color
color r=117 g=91 b=73
>>> newcolor=pickAColor()
>>> print newcolor
color r=255 g=51 b=51
```

Change Colors Directly

```
>>> file="C:/mediasources/barbara.jpg"
>>> pict=makePicture(file)
>>> show(pict)
>>> setColor(getPixel(pict,10,100),yellow)
>>> setColor(getPixel(pict,11,100),yellow)
>>> setColor(getPixel(pict,12,100),yellow)
>>> setColor(getPixel(pict,13,100),yellow)
>>> repaint(pict)
```

LAB 3

- In JES Command area,
  - Select a '.jpg' file by pickAFile()
  - Convert the selected file to a picture file
  - Change any ten adjacent pixels into red color

Can Change Colors in Pixels

- We did

```
>>> setColor(getPixel(pict,10,100),yellow)
>>> setColor(getPixel(pict,11,100),yellow)
>>> setColor(getPixel(pict,12,100),yellow)
>>> setColor(getPixel(pict,13,100),yellow)
```

- This is the same as

```
>>> setColor(getPixel(pict,10,100),yellow)
>>> setColor(getPixel(pict,10+1,100),yellow)
>>> setColor(getPixel(pict,10+2,100),yellow)
>>> setColor(getPixel(pict,10+3,100),yellow)
```

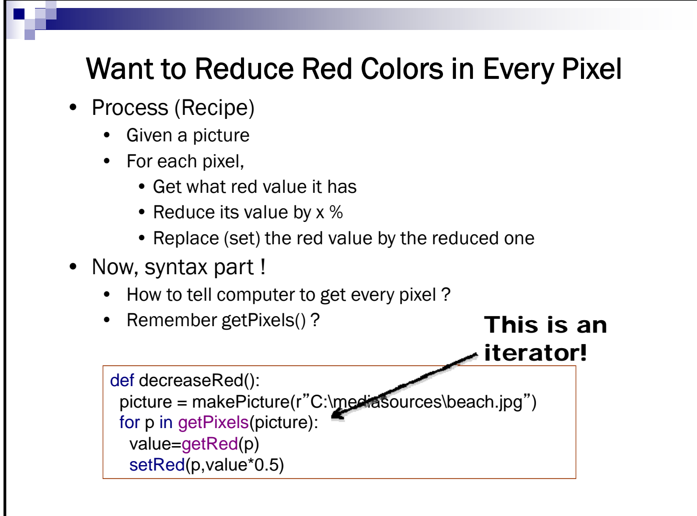
Any Way to Automate This ?

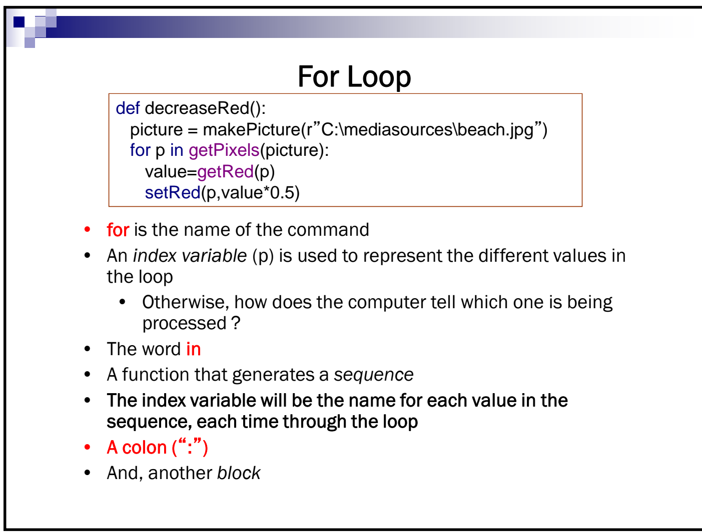
- We have

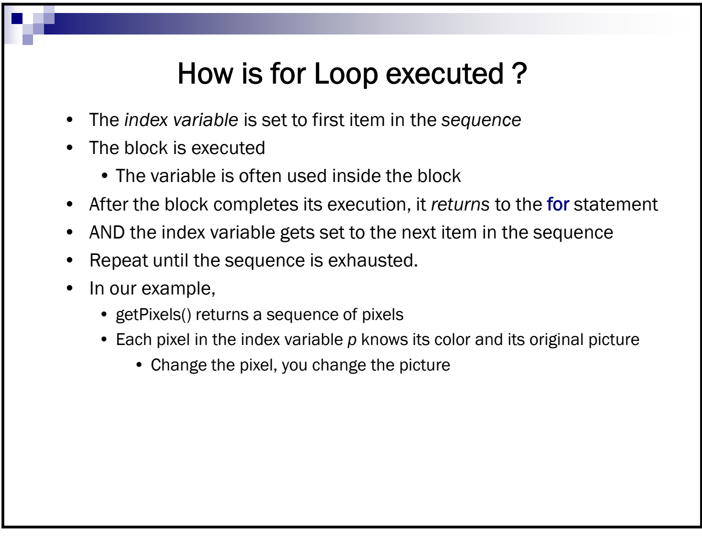
```
>>> setColor(getPixel(pict,10,100),yellow)
>>> setColor(getPixel(pict,10+1,100),yellow)
>>> setColor(getPixel(pict,10+2,100),yellow)
>>> setColor(getPixel(pict,10+3,100),yellow)
```

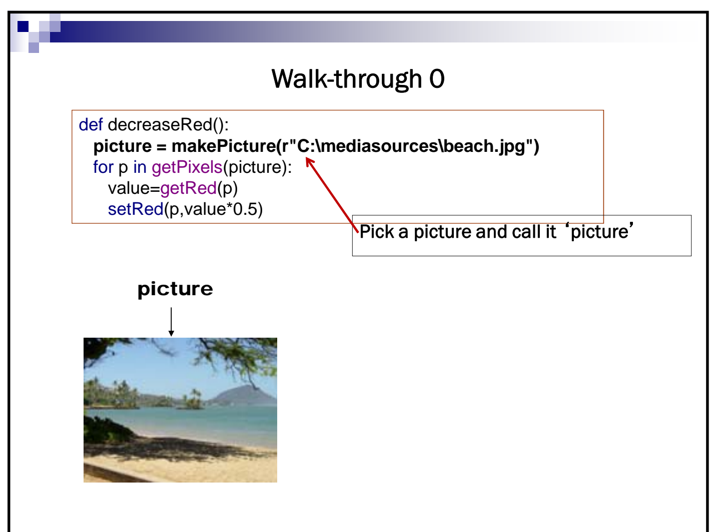
- Problem solving – like a cooking recipe, a manual, …
  - Get an overall idea of how to approach the problem => PROCESS
  - Specify the steps to go through to solve the problem
  - Use the syntax in the given programming language
- "for var in range(0, 4):"

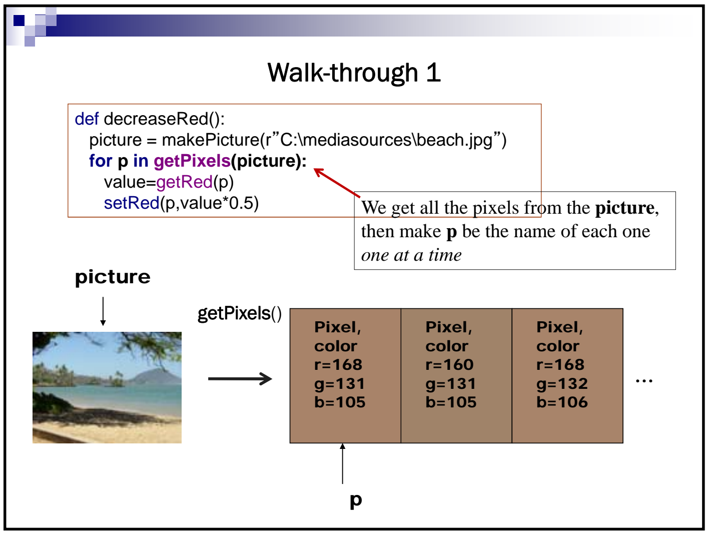
```
>>> for i in range(4):
... setColor(getPixel(pict,10+i,100),yellow)
```


Want to Reduce Red Colors in Every Pixel
Process (Recipe)
Given a picture
For each pixel,
Get what red value it has
Reduce its value by x %
Replace (set) the red value by the reduced one
Now, syntax part !
How to tell computer to get every pixel ?
Remember getPixels() ?
This is an iterator!
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)

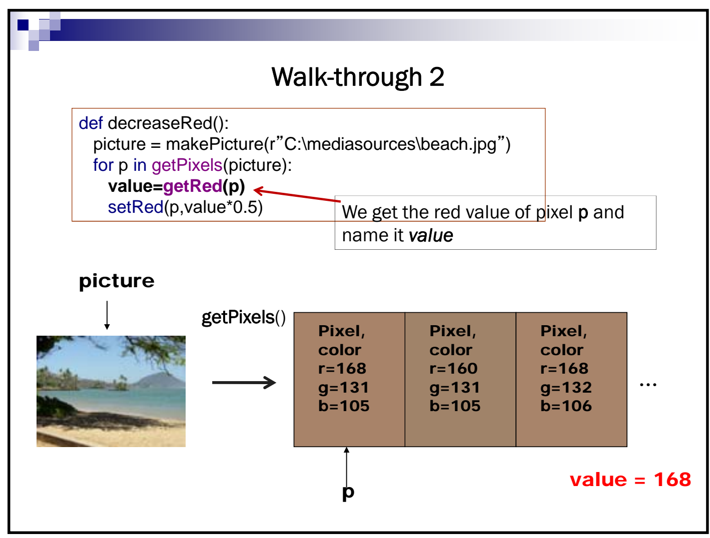

For Loop
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
for is the name of the command
An index variable (p) is used to represent the different values in the loop
Otherwise, how does the computer tell which one is being processed ?
The word in
A function that generates a sequence
The index variable will be the name for each value in the sequence, each time through the loop
A colon (":")
And, another block


How is for Loop executed ?
The index variable is set to first item in the sequence
The block is executed
The variable is often used inside the block
After the block completes its execution, it returns to the for statement
AND the index variable gets set to the next item in the sequence
Repeat until the sequence is exhausted.
In our example,
getPixels() returns a sequence of pixels
Each pixel in the index variable p knows its color and its original picture
Change the pixel, you change the picture

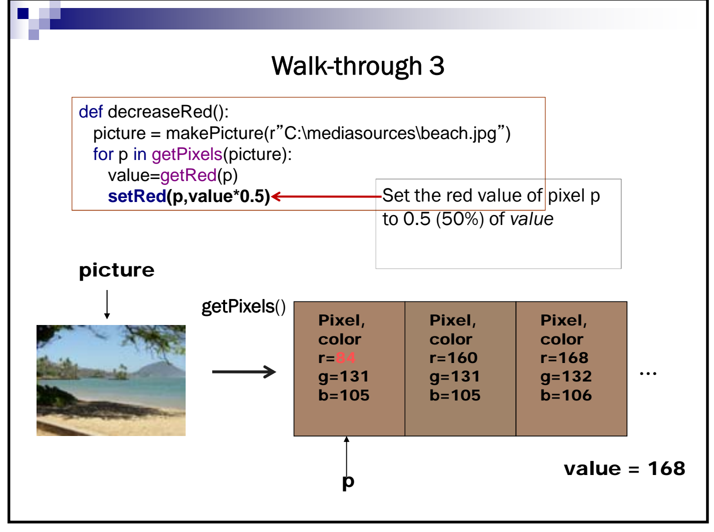

Walk-through 0
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
Pick a picture and call it 'picture'
picture



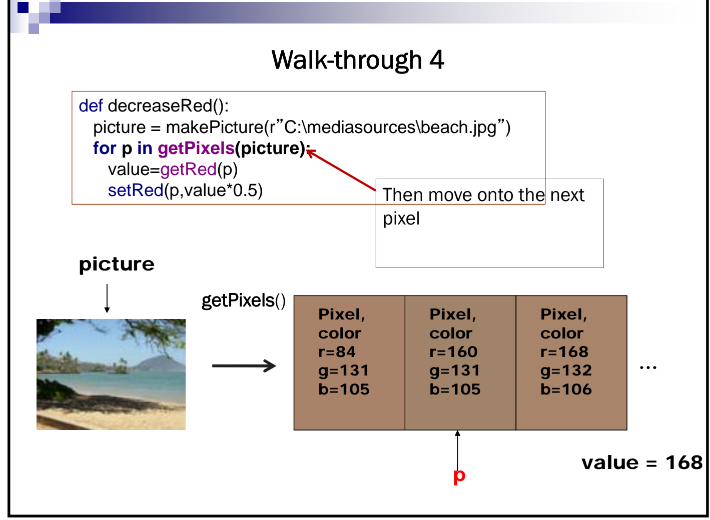
Walk-through 1
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
We get all the pixels from the picture, then make p be the name of each one one at a time
picture
getPixels()
Pixel, color r=168 g=131 b=105
Pixel, color r=160 g=131 b=105
Pixel, color r=168 g=132 b=106
…
p



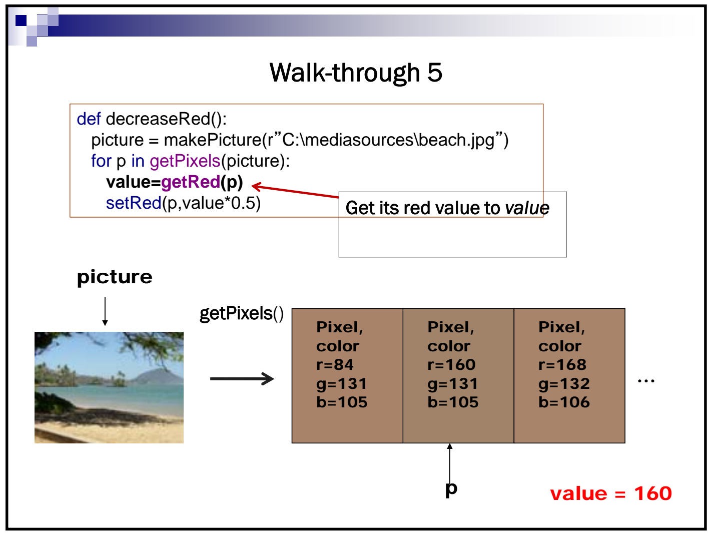
Walk-through 2
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
We get the red value of pixel p and name it value
picture
getPixels()
Pixel, color r=168 g=131 b=105
Pixel, color r=160 g=131 b=105
Pixel, color r=168 g=132 b=106
…
p
value = 168

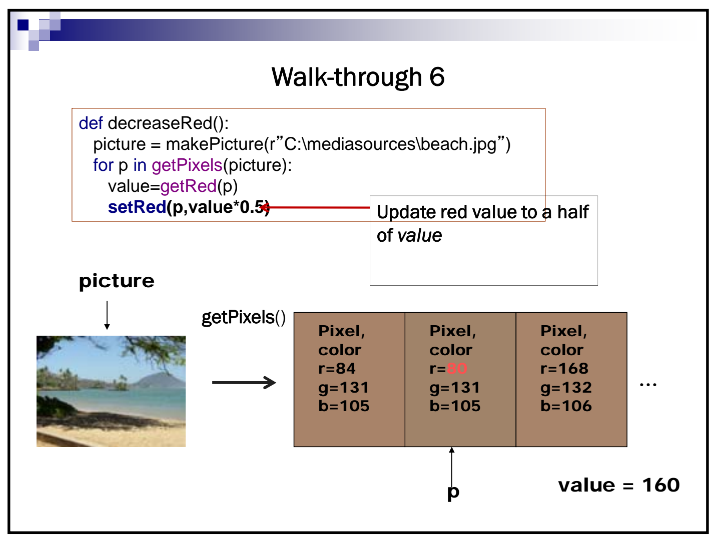


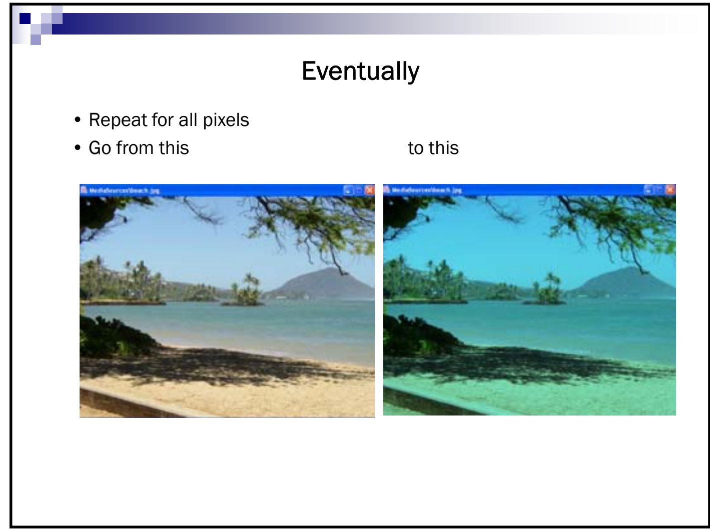
Walk-through 3
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
Set the red value of pixel p to 0.5 (50%) of value
picture
getPixels()
Pixel, color r=84 g=131 b=105
Pixel, color r=160 g=131 b=105
Pixel, color r=168 g=132 b=106
…
p
value = 168

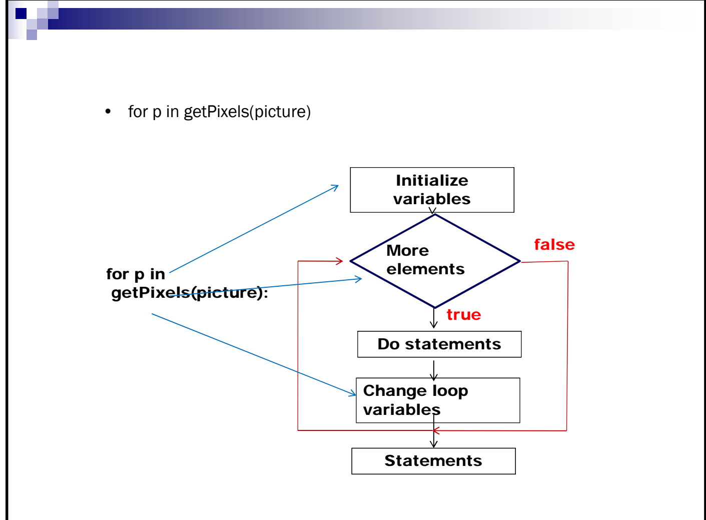


Walk-through 4
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
Then move onto the next pixel
picture
getPixels()
Pixel, color r=84 g=131 b=105
Pixel, color r=160 g=131 b=105
Pixel, color r=168 g=132 b=106
…
p
value = 168


Walk-through 5
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
Get its red value to value
picture
getPixels()
Pixel, color r=84 g=131 b=105
Pixel, color r=160 g=131 b=105
Pixel, color r=168 g=132 b=106
…
p
value = 160


Walk-through 6
def decreaseRed():
picture = makePicture(r"C:\mediasources\beach.jpg")
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
Update red value to a half of value
picture
getPixels()
Pixel, color r=84 g=131 b=105
Pixel, color r=80 g=131 b=105
Pixel, color r=168 g=132 b=106
…
p
value = 160


Eventually
Repeat for all pixels
Go from this
to this


for p in getPixels(picture)
Initialize variables
More elements
false
true
for p in getPixels(picture):
Do statements
Change loop variables
Statements

Tracing/Stepping/Walking Through

- What we just did is called "stepping" or "walking through" the program
  - You consider each step of the program, in the order that the computer would execute it
  - You consider what would *specifically* happen there
  - You write down what values each variable (name) has at each point.
- It's one of the most important *debugging* skills you can have.
  - And *everyone* has to do a *lot* of debugging, especially at first.

LAB 4

- In JES Program area,
  - Write a function, increaseRed(picFile)
    - Increases a red value in every pixel in the picture file, picFile, by 30%
- And run the new function increaseRed(picFile) in the Command area

Make decreaseRed() More General ?

```
def decreaseRed():
  picture = makePicture(r"C:\mediasources\beach.jpg")
  for p in getPixels(picture):
    value=getRed(p)
    setRed(p,value*0.5)
```

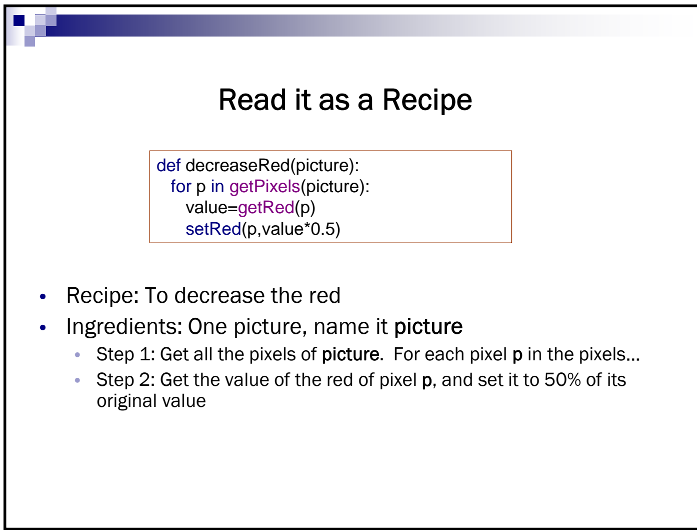
- decreaseRed() works only for beach picture
- How to generalize it ?
  - Decrease red color can be applied to any picture
  - Remove hard-coding
  - Have the picture be specified

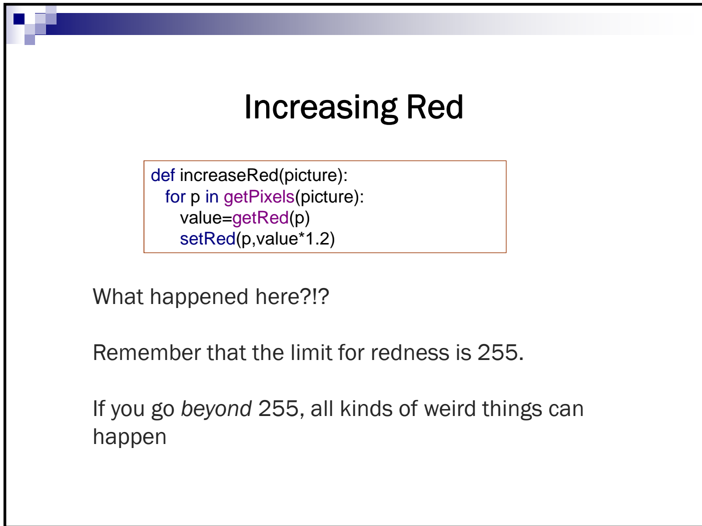
```
def decreaseRed(picture):
  for p in getPixels(picture):
    value=getRed(p)
    setRed(p,value*0.5)
```

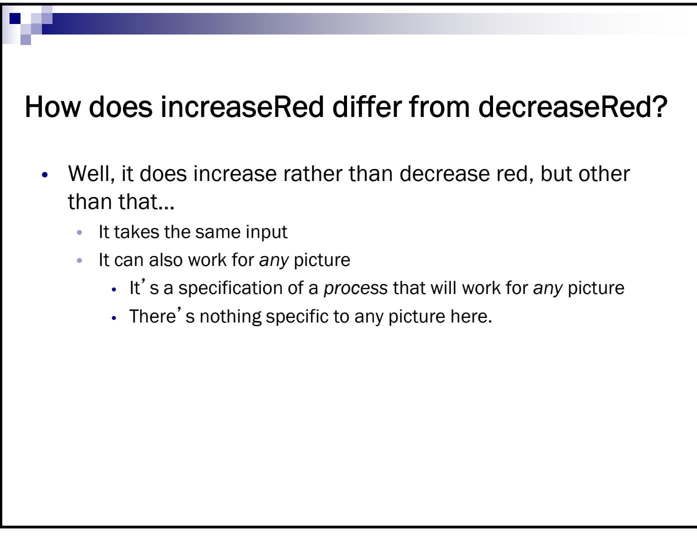
Did It Really Work ?

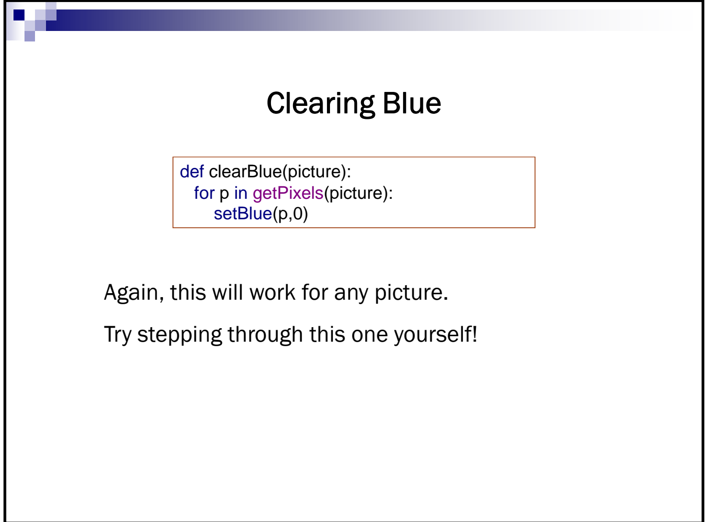
- How can we be sure ?
  - Sure, the picture looks different
  - But, did we indeed decrease the amount of red ?
  - And by the exact amount we want ?
- We can check a few pixels as on the right

```
>>> file = pickAFile()
>>> print file
C:\mediasources\beach.jpg
>>> pict = makePicture(file)

# check one pixel values
>>> pixel = getPixel(pict,1,1)
>>> print pixel
Pixel, color=color r=168 g=131 b=105
>>> decreaseRed(pict)
>>> print pixel
Pixel, color=color r=168 g=131 b=105

# check updated values of the same pixel
>>> newPixel = getPixel(pict,1,1)
>>> print newPixel
Pixel, color=color r=84 g=131 b=105
>>> print 168 * 0.5
84.0
```


Read it as a Recipe
def decreaseRed(picture):
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)
Recipe: To decrease the red
Ingredients: One picture, name it picture
Step 1: Get all the pixels of picture. For each pixel p in the pixels…
Step 2: Get the value of the red of pixel p, and set it to 50% of its original value


Increasing Red
def increaseRed(picture):
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*1.2)
What happened here?!?
Remember that the limit for redness is 255.
If you go beyond 255, all kinds of weird things can happen

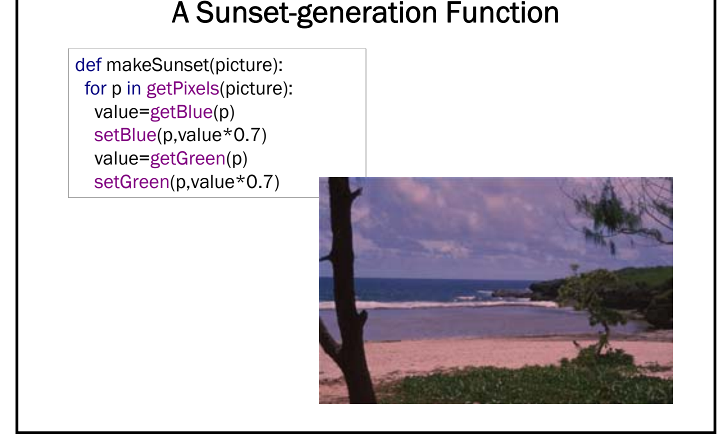

How does increaseRed differ from decreaseRed?
Well, it does increase rather than decrease red, but other than that…
It takes the same input
It can also work for any picture
It's a specification of a process that will work for any picture
There's nothing specific to any picture here.


Clearing Blue
def clearBlue(picture):
for p in getPixels(picture):
setBlue(p,0)
Again, this will work for any picture.
Try stepping through this one yourself!

Can we combine these?

- How do we turn this beach scene into a sunset?
- What happens at sunset?
  - At first, try increasing the red
  - but that made things like red specks in the sand REALLY prominent.
    - That can't be how it really works
  - New Theory:
    - As the sun sets, less blue and green is visible
    - This makes things look more red.

A Sunset-generation Function

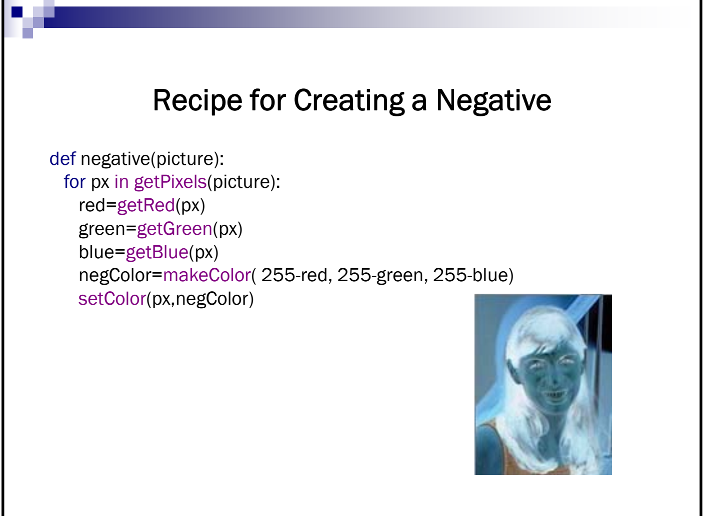


```
def makeSunset(picture):
 for p in getPixels(picture):
   value=getBlue(p)
   setBlue(p,value*0.7)
   value=getGreen(p)
   setGreen(p,value*0.7)
```

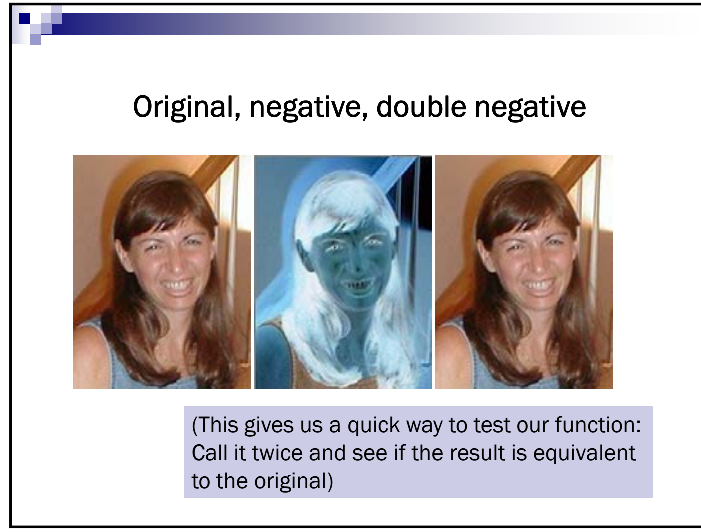
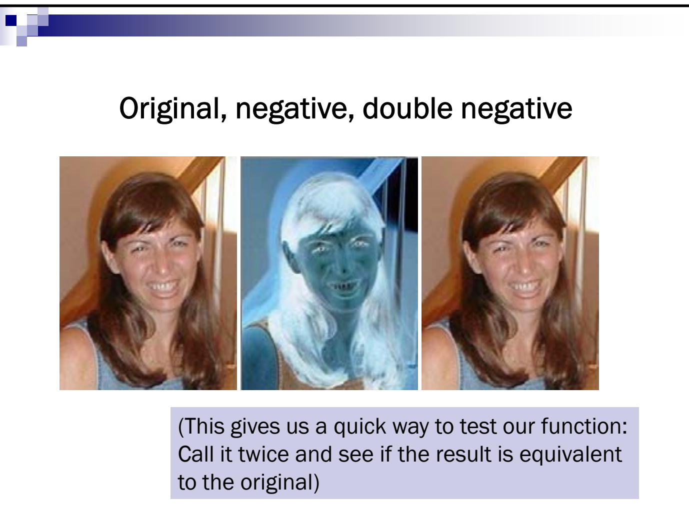
Creating a Negative

- What is a negative ?
  - R,G,B go from 0 to 255
  - Let's say Red is 10. That's very light red.
    - What's the opposite? LOTS of Red!
  - The negative of that would be 245: 255-10
- Process
  - For each pixel,
    - negate each color component in creating a new color
  - Then, we negate the whole picture.

Recipe for Creating a Negative



```
def negative(picture):
  for px in getPixels(picture):
    red=getRed(px)
    green=getGreen(px)
    blue=getBlue(px)
    negColor=makeColor( 255-red, 255-green, 255-blue)
    setColor(px,negColor)
```

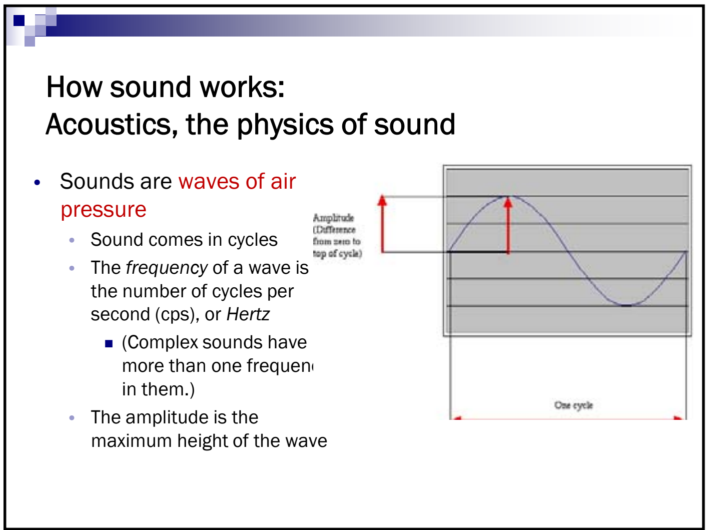
Converting to Grayscale

- We know that if red=green=blue, we get gray
  - But what value do we set all three to?
- What we need is a value representing the darkness of the color, the *luminance*
- There are lots of ways of getting it, but one way that works reasonably well is dirt simple—simply take the average:

$$\frac{(red+green+blue)}{3}$$

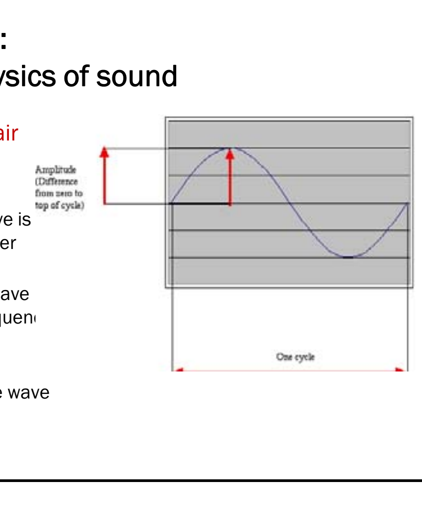
LAB 5

- In JES Program area,
  - Write a function to create a negative or a grayscale of a picture
- And run the new function in the Command area



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 frequen in them.)
  - The amplitude is the maximum height of the wave

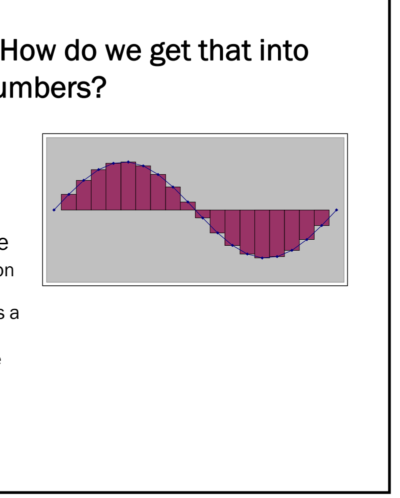


Volume and pitch:

- 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 can hear between 5 Hz and 20,000 Hz (20 kHz)
  - A above middle C is 440 Hz

Digitizing 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?

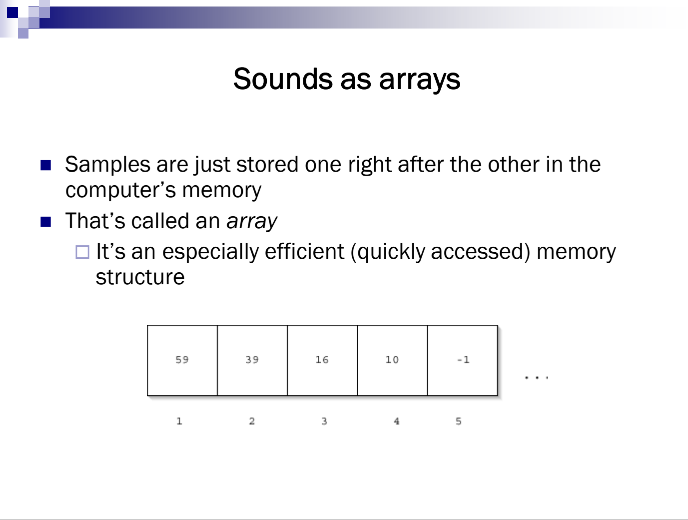


Nyquist Theorem

- We need twice as many samples as the maximum frequency in order to represent (and recreate, later) the original sound.
- The number 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 phones 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

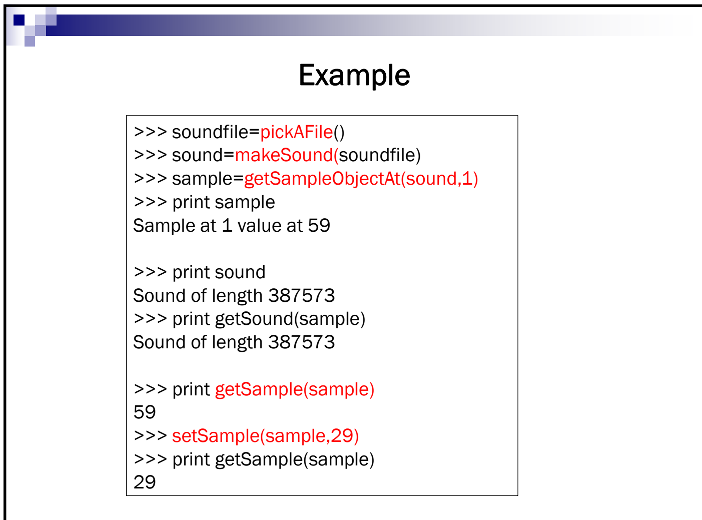
Digitizing Sound

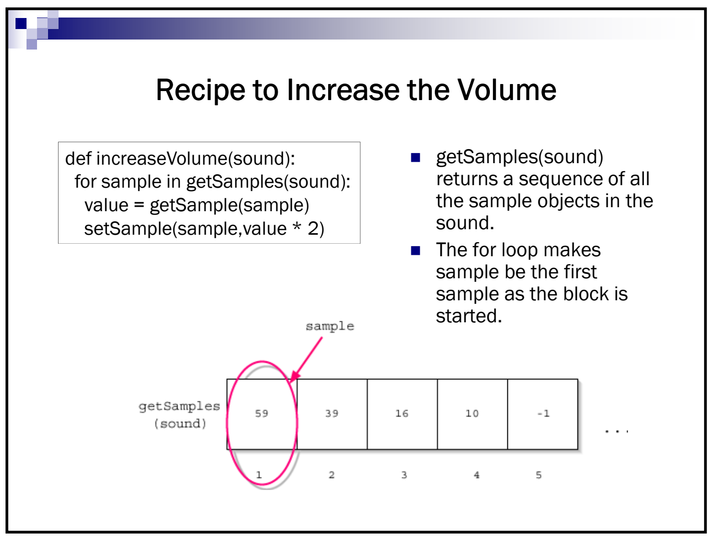
- Each sample is stored as a number (two bytes)
- What's the range of available combinations?
  - 16 bits, 2^{16} = 65,536
  - But we want both positive and negative values
  - What if we use one bit to indicate positive (0) or negative (1)?
  - That leaves us with 15 bits
  - 15 bits, 2^{15} = 32,768
  - One of those combinations will stand for zero
    - We'll use a "positive" one, so that's one less pattern for positives
- Each sample can be between -32,768 and 32,767

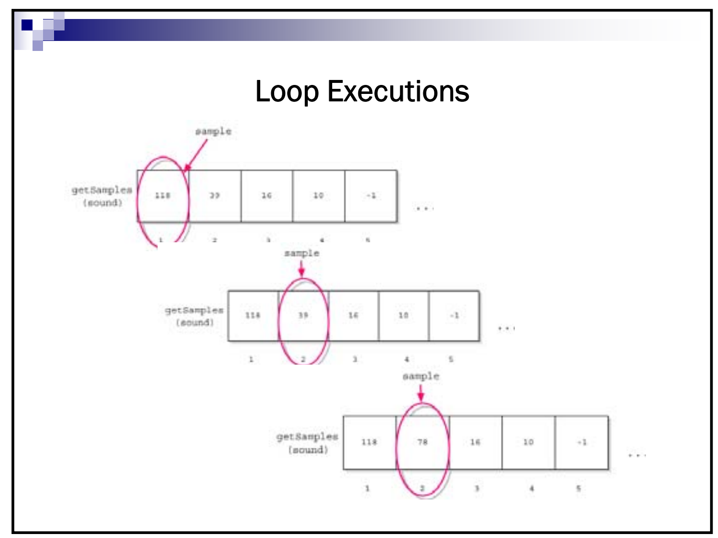


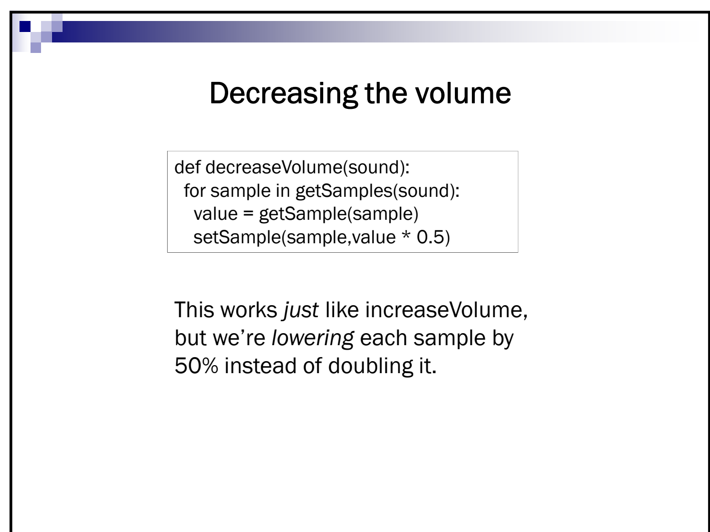
Working with sounds

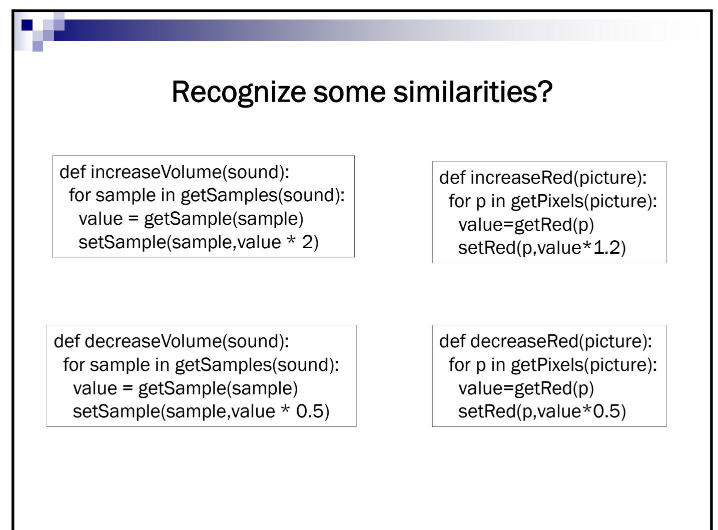
- We'll use pickAFile and makeSound as we have before.
  - But now we want .wav files
- We'll use getSamples to get all the *sample objects* out of a sound
- We can also get the value at any index with getSampleValueAt(sound, index)
- A sample object remembers its sound, so if you change the sample object, the sound gets changed by
  - setSample(sample,value)
- Can save sounds with writeSoundTo(sound,"file.wav")

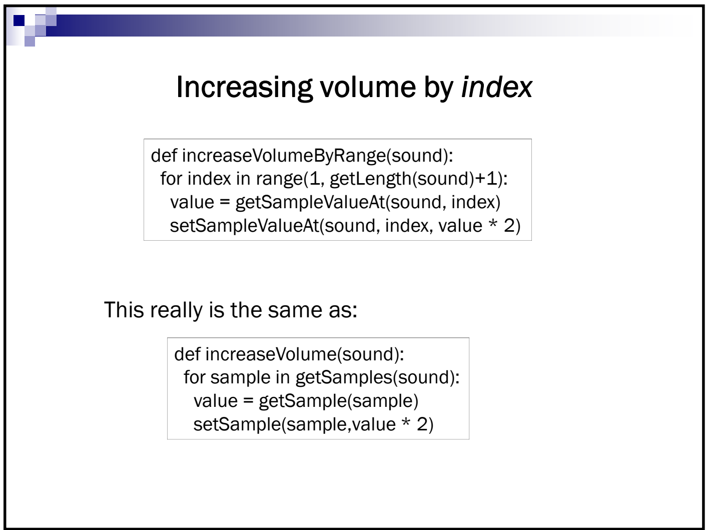


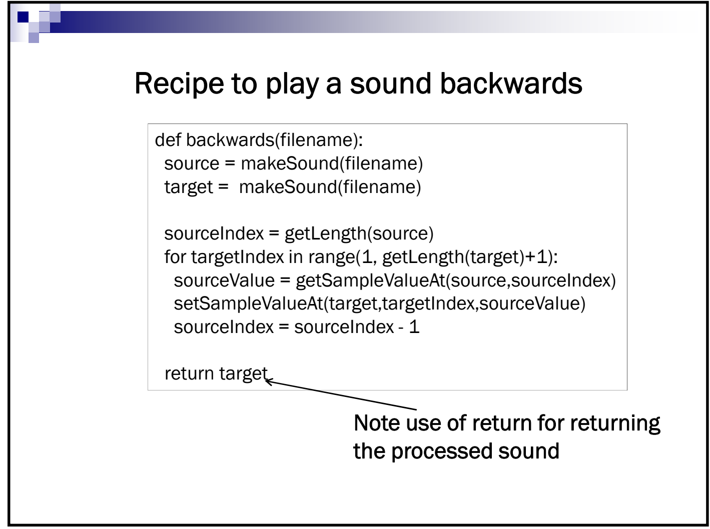

Recipe to Increase the Volume
def increaseVolume(sound):
for sample in getSamples(sound):
value = getSample(sample)
setSample(sample,value * 2)
getSamples(sound) returns a sequence of all the sample objects in the sound.
The for loop makes sample be the first sample as the block is started.
sample
getSamples (sound)
59
39
16
10
-1
1
2
3
4
5

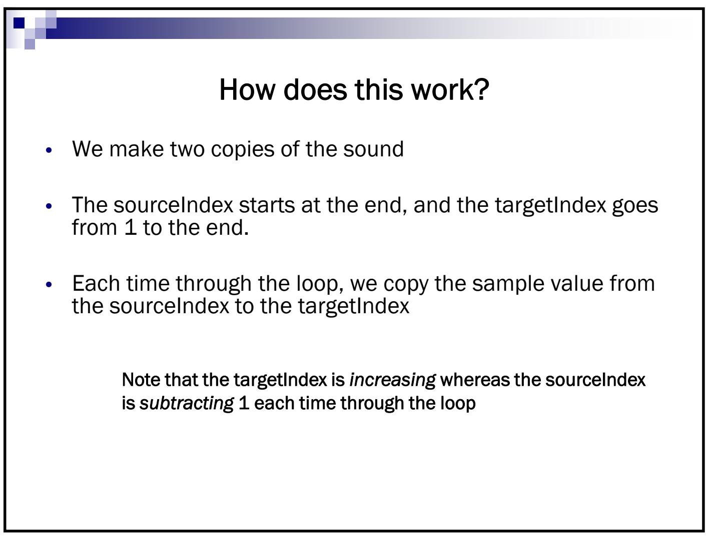

Loop Executions
sample
getSamples (sound)

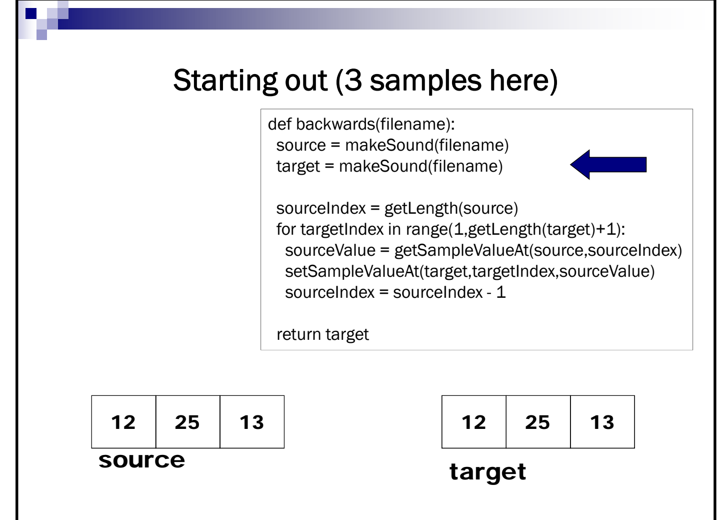

Decreasing the volume
def decreaseVolume(sound):
for sample in getSamples(sound):
value = getSample(sample)
setSample(sample,value * 0.5)
This works just like increaseVolume, but we're lowering each sample by 50% instead of doubling it.

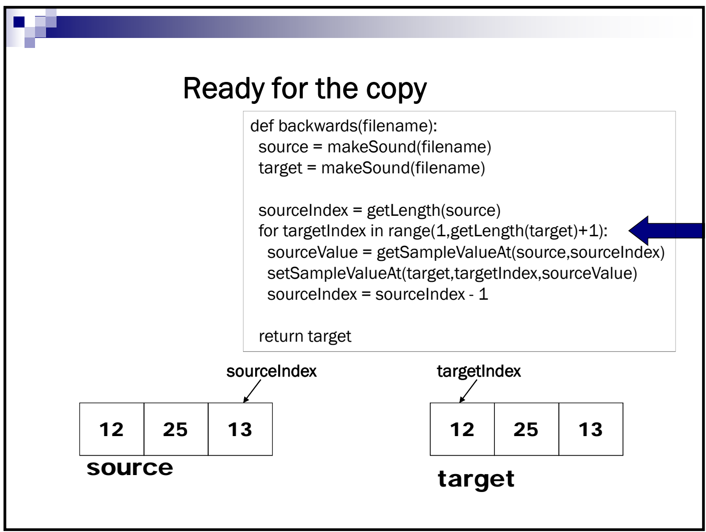

Recognize some similarities?
def increaseVolume(sound):
for sample in getSamples(sound):
value = getSample(sample)
setSample(sample,value * 2)
def increaseRed(picture):
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*1.2)
def decreaseVolume(sound):
for sample in getSamples(sound):
value = getSample(sample)
setSample(sample,value * 0.5)
def decreaseRed(picture):
for p in getPixels(picture):
value=getRed(p)
setRed(p,value*0.5)

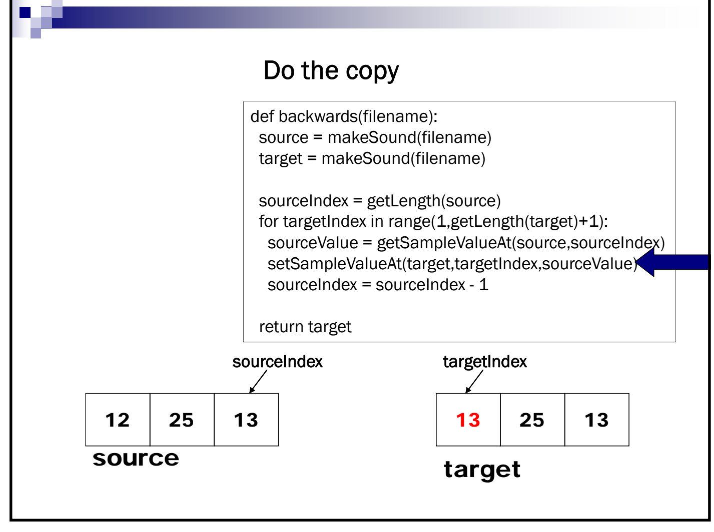

Increasing volume by *index*
def increaseVolumeByRange(sound):
 for index in range(1, getLength(sound)+1):
  value = getSampleValueAt(sound, index)
  setSampleValueAt(sound, index, value * 2)
This really is the same as:
def increaseVolume(sound):
 for sample in getSamples(sound):
  value = getSample(sample)
  setSample(sample,value * 2)

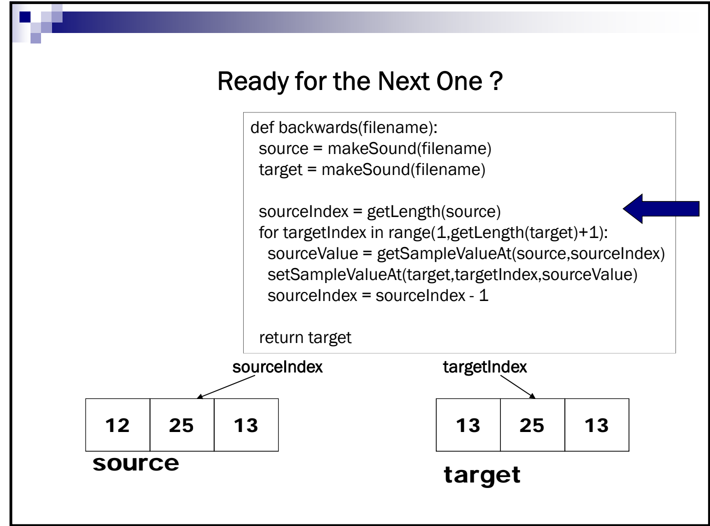

Recipe to play a sound backwards
def backwards(filename):
 source = makeSound(filename)
 target = makeSound(filename)
 sourceIndex = getLength(source)
 for targetIndex in range(1, getLength(target)+1):
  sourceValue = getSampleValueAt(source,sourceIndex)
  setSampleValueAt(target,targetIndex,sourceValue)
  sourceIndex = sourceIndex - 1
 return target
Note use of return for returning the processed sound

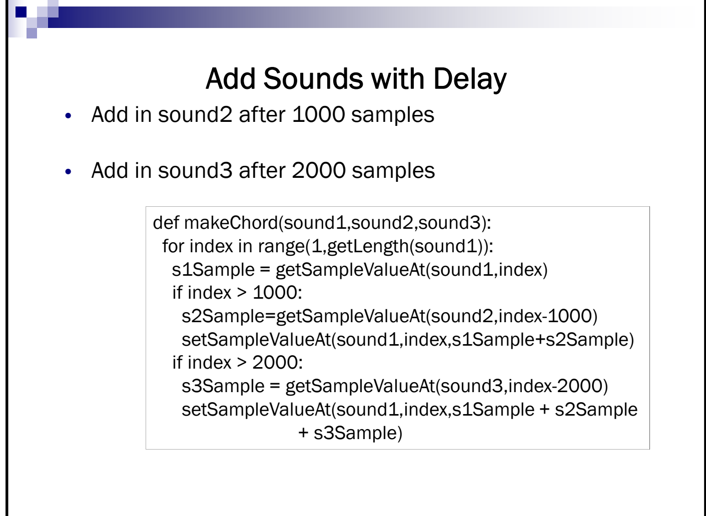

How does this work?
We make two copies of the sound
The sourceIndex starts at the end, and the targetIndex goes from 1 to the end.
Each time through the loop, we copy the sample value from the sourceIndex to the targetIndex
Note that the targetIndex is *increasing* whereas the sourceIndex is *subtracting* 1 each time through the loop

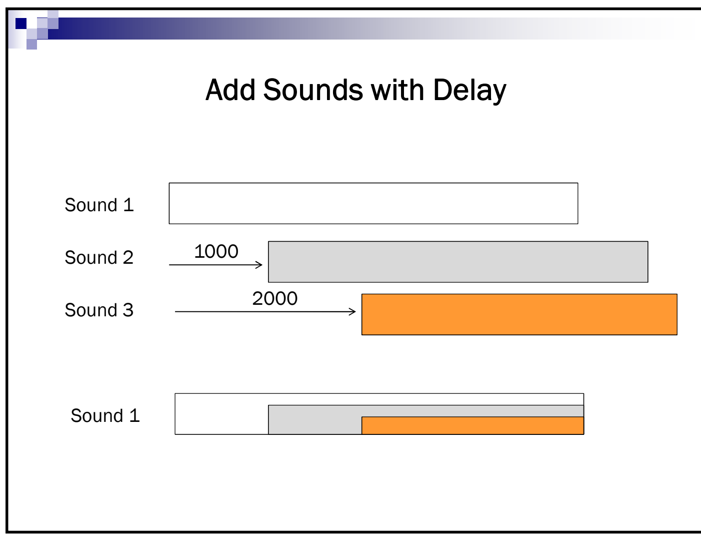

Starting out (3 samples here)
def backwards(filename):
 source = makeSound(filename)
 target = makeSound(filename)
 sourceIndex = getLength(source)
 for targetIndex in range(1,getLength(target)+1):
  sourceValue = getSampleValueAt(source,sourceIndex)
  setSampleValueAt(target,targetIndex,sourceValue)
  sourceIndex = sourceIndex - 1
 return target
12
25
13
source
12
25
13
target

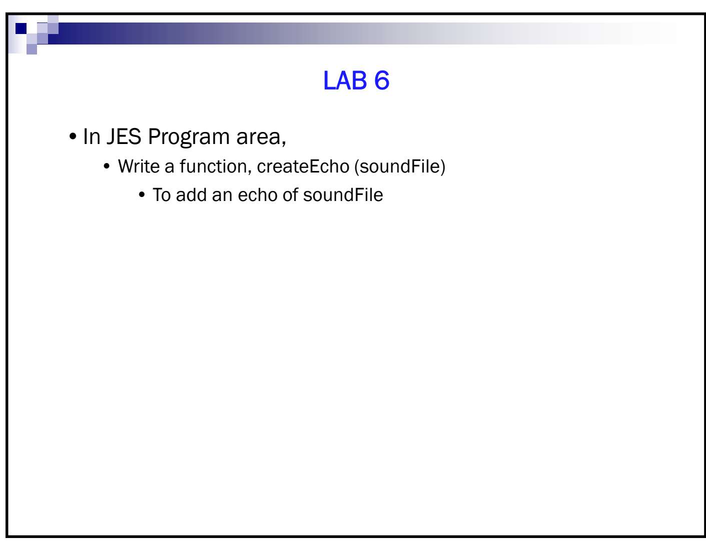

Ready for the copy
def backwards(filename):
source = makeSound(filename)
target = makeSound(filename)
sourceIndex = getLength(source)
for targetIndex in range(1,getLength(target)+1):
sourceValue = getSampleValueAt(source,sourceIndex)
setSampleValueAt(target,targetIndex,sourceValue)
sourceIndex = sourceIndex - 1
return target
sourceIndex
targetIndex
12
25
13
source
12
25
13
target


Do the copy
def backwards(filename):
source = makeSound(filename)
target = makeSound(filename)
sourceIndex = getLength(source)
for targetIndex in range(1,getLength(target)+1):
sourceValue = getSampleValueAt(source,sourceIndex)
setSampleValueAt(target,targetIndex,sourceValue)
sourceIndex = sourceIndex - 1
return target
sourceIndex
targetIndex
12
25
13
source
13
25
13
target


Ready for the Next One ?
def backwards(filename):
source = makeSound(filename)
target = makeSound(filename)
sourceIndex = getLength(source)
for targetIndex in range(1,getLength(target)+1):
sourceValue = getSampleValueAt(source,sourceIndex)
setSampleValueAt(target,targetIndex,sourceValue)
sourceIndex = sourceIndex - 1
return target
sourceIndex
targetIndex
12
25
13
source
13
25
13
target


Add Sounds with Delay
Add in sound2 after 1000 samples
Add in sound3 after 2000 samples
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 Sounds with Delay
Sound 1
Sound 2
1000
Sound 3
2000
Sound 1


LAB 6
• In JES Program area,
• Write a function, createEcho (soundFile)
• To add an echo of soundFile