Increasing volume by sample index

We've done this…

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

This does the same thing, but more flexible…

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

Knowing where we are in the sound

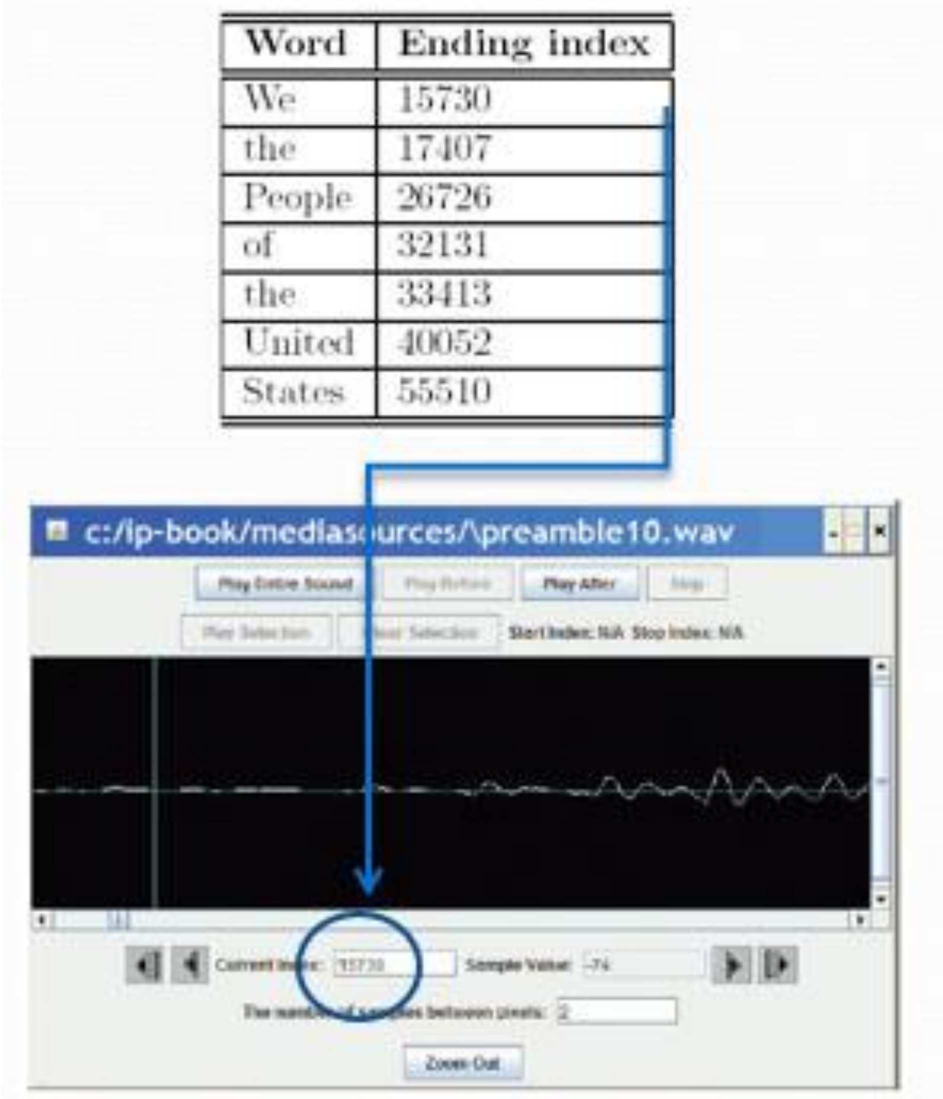
- More complex operations require us to know where we are in the sound, which sample
  - Not just process all the samples exactly the same
- Examples:
  - **Splicing/Merging** sounds
  - **Reversing** a sound
    - It's just copying, like we did with pixels
  - **Changing the frequency** of a sound
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

Splicing Sounds

- Splicing gets its name from literally cutting and pasting pieces of magnetic tape together
- Doing it digitally is easy in principle
- Algorithm:
  - take two sound files
  - merge them into one sound file
  - play them one after the other with a one second break in-between

Finding the Word End-Points

- Using MediaTools and play before/after cursor, we can figure out the index numbers where each word ends
- We want to splice a copy of the word “United” after “We the” so that it says, “We the United People of the United States”.



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

c:/ip-book/mediasources/\preamble10.wav

Start Index: N/A Stop Index: N/A

Current Index: 15730

Sample Value: -74

The number of samples between pixels: 2

Abstractualization

General Clip Function

- Simplify splicing functions using a method that takes a start and end index, and then returns a new sound clip with just that part of the original sound in it.

```
def clip(source, start, end):
  target = makeEmptySound(end - start)
  tIndex = 0
  for sIndex in range(start, end):
    value = getSampleValueAt(source, sIndex)
    setSampleValueAt(target, tIndex, value)
    tIndex = tIndex + 1
  return target
```

Abstractualization

General Copy Function

- Simplify splicing with a general copy method that takes source and target sounds, then copies the source into the target starting at a specified location

```
def copy(source, target, start):
  tIndex = start
  for sIndex in range(0, getLength(source)):
    value = getSampleValueAt(source, sIndex)
    setSampleValueAt(target, tIndex, value)
    tIndex = tIndex + 1
```

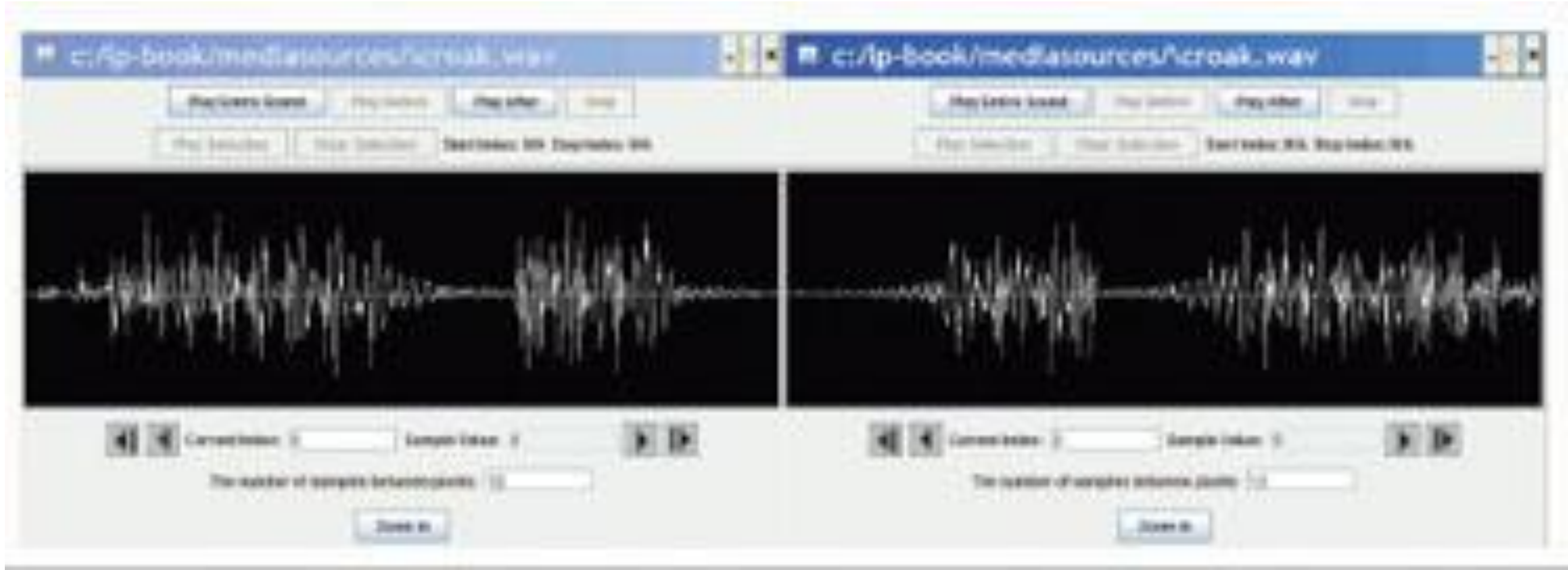
Example

Simplified Preamble Splice

```
def createNewPreamble():
  file = getMediaPath("preamble10.wav")
  preamble = makeSound(file)           # old preamble
  united = clip(preamble, 33414, 40052) # "United"
  start = clip(preamble, 0, 17407)      # "We the"
  end = clip(preamble, 17408, 55510)    # the rest
  len = getLength(start) + getLength(united)
  len = len + getLength(end)  # length of everything
  newPre = makeEmptySound(len)         # new preamble
  copy(start, newPre, 0)
  copy(united, newPre, getLength(start))
  copy(end, newPre, getLength(start)+getLength(united))
  return newPre
```

Reversing Sounds

- We can modify sounds by reversing them
- Algorithm…
  - Get each sample from a source sound, starting at the end
  - Write each sample from the source into a new, empty sound, starting at the beginning


c:/ip-book/mediasources/croak.wav
c:/ip-book/mediasources/croak.wav

Reversing Sounds

Pseudo code…

- Start with a sound (object)
- Make an new, empty sound, the same length
- Start a counter; set it to the number of the last (highest) index from the source
- For each index in the target
  - Get the sample value of the source, starting at the last
  - Set the sample value of the target using the value from the source
  - Increment the counter, minus one
- Return the target to the system

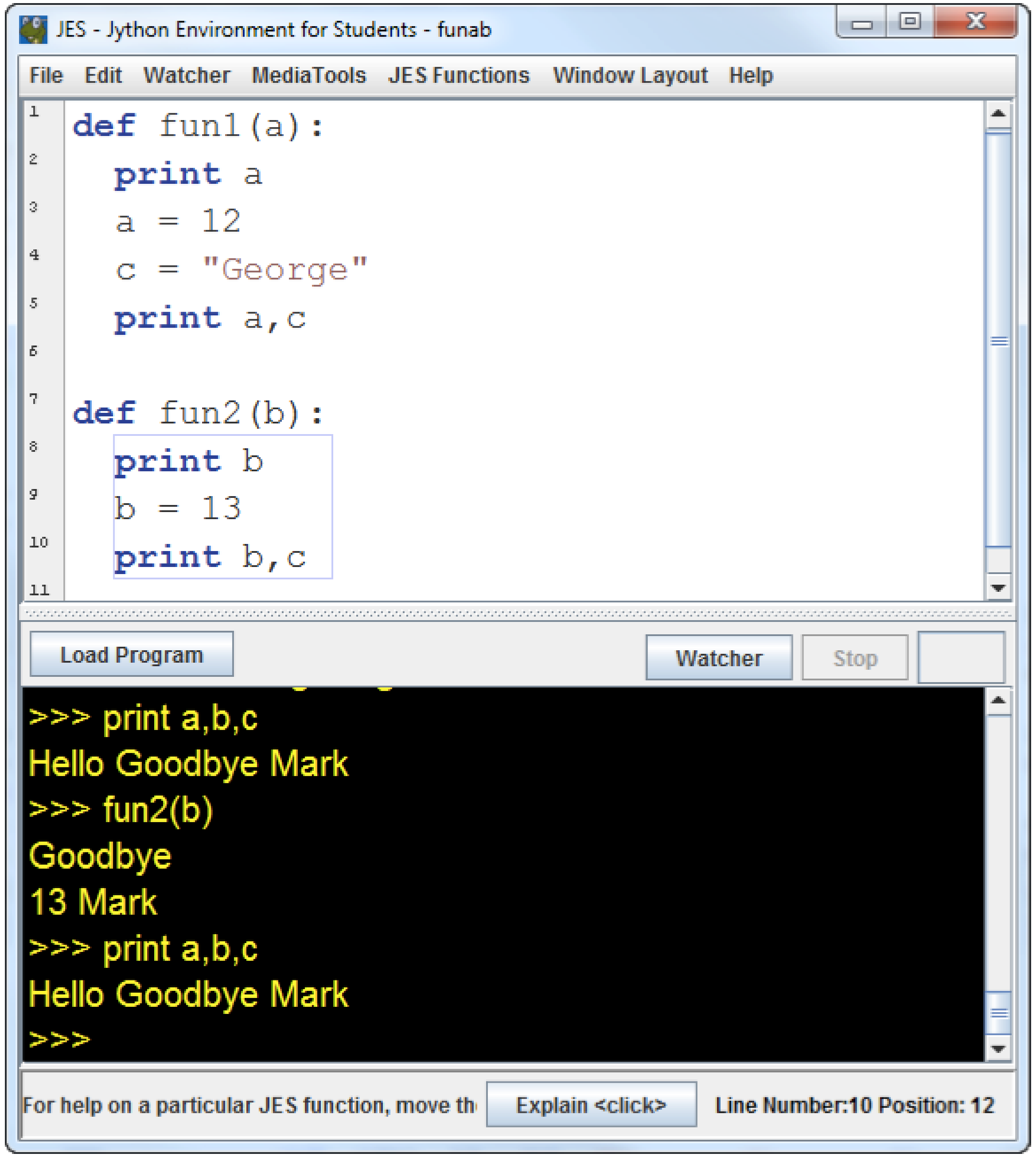
Beware: when *making an empty sound*, need to match the sample rate of the source!

- Use `getSampleRate(sourceObj)`
- `makeEmptySound(len(sourceObj), 44100)`

Functions and Scope

- Defined:
  - Let's call the variable that represents the input a “parameter variable”
- Key idea:
  - The parameter variable in a function has *NOTHING* to do with any variable (even with the same name) in the Command Area – or anywhere else.
- Parameter variables are *LOCAL* to the function.
  - We say that it's in the function's *SCOPE*.

Think this through:


JES - Jython Environment for Students - funab
File Edit Watcher MediaTools JES Functions Window Layout Help
Load Program
Watcher
Stop

```
def fun1(a):
  print a
  a = 12
  c = "George"
  print a,c

def fun2(b):
  print b
  b = 13
  print b,c
```

```
>>> print a,b,c
Hello Goodbye Mark
>>> fun2(b)
Goodbye
13 Mark
>>> print a,b,c
Hello Goodbye Mark
>>>
```

For help on a particular JES function, move th
Explain <click>
Line Number:10 Position: 12

Values are copied into parameters

- When a function is called, the input values are copied into the parameter variables.
    - Changing the parameter variables <u>*can't*</u> change the input variables.
- All variables that are local disappear at the end of the function.
- We can reference variables external to the function, if we don't have a local variable with the same name.

Parameters as Objects

- ***Note:*** Slightly different when you pass an object, like a Sound or a Picture.
  - You still can't change the original *variable*, but you've passed in the object. You can change the object.

```
>>> p = makePicture(pickAFile())
>>> increaseRed(p)
```

- increaseRed() can't change the variable **p**, but it can apply functions and methods to change the ***picture*** that **p** references.
- That picture, the object, is the *value* that we passed in to the function.