CS116 – List indexing - Part 2

Purpose: We don't always want to process an entire list A common schema is to look at a chunk of a list by giving an *offset* which is the starting position in the list and then using an index added to that offset.

Schema: process a chunk of a list using *offset* and *index*

```
for index in range(0,length of chunk):
      process list[offset+index]
```

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 30 minutes.

1. The following Sound method sets a chunk of the given *length* of the sound to silence starting at position $offset$.

   ```
   def silenceStartingAt(self,offset,length):
         samples = self.getSamples()
         for index in range(0, length):
             samples[offset+index].setValue(0)
   ```

   If the offset is 1000 and the length of the chunk is 500, what is the first sample that is set to 0?

   What is the next sample that is set to 0?

   Which sample positions will be set to 0?

2. The following Picture method copies the chunk of the first half of a picture to the second half of the picture.

   ```
   def repeatFirstHalf(self):
       pixels = self.getPixels()
       offset = len(pixels)//2
       for i in range(0,len(pixels)//2):
           pixels[offset+i].setColor(pixels[i].getColor())
   ```

   If the picture has 1000 pixels what is the first pixel that will be altered?

   That first pixel will get its new color from which pixel position?

   Which pixel positions will be changed?

3. What minor changes to the `repeatFirstHalf` method would make it copy the second quarter of the picture to the third quarter of the picture (leaving the first and fourth quarters of the picture unchanged).

Complete each of the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies or if you get an error message that you do not understand.
For list indexing problems it will help if you ask yourself the following questions:

1. What are the offset(s) of any list chunks?
2. What range is needed for the index?
3. How should elements in the chunks be processed?

Assignment 1:
Write Sound method taperEnd(self,num) that returns a new sound that is the same as the original sound except that the last num samples taper off to zero volume at a constant rate Hint: Consider the ratio $(num - i)/num$. When i is 0 this ratio will be 1 and when i is num it will be 0 so this will give you your taper factor.

Criteria of Success: Take a long sound like any of the bassoon files and use the explorer to look at the sound wave. There are over 50,000 sound samples in that file and most of the waves should all be the same height. If you explore the result of tapering the last 20,000 samples you should be able to clearly see the sound wave getting smaller, and smaller until it tapers down to zero at the end. Play the sound and verify that the volume tapers off at the end.

Assignment 2:
We can combine two sounds together by simply adding their values. Write a Sound method `overlay(self,background,start)` that overlays the original sound on top of a background sound at the given starting sample location. Your method should return this new resulting sound.

You may assume that the background sound is long enough so that the original sound will fit entirely over the background.

Criteria of Success: When you play the resulting sounds you should hear both sounds simultaneously during the overlayed portion.

Assignment 3:
Write a Sound method `copyChunk(self,newSound,offset1,offset2,length)` This method should take a chunk from the original sound starting at offset1 and the given length. This chunk should be copied to the newSound starting at offset2.

You may assume that the newSound object is long enough for the copying.

Criteria of Success: When you play the newSound you should hear the copied chunk.

Assignment 4:
Write a Sound method `stutter(offset,length, numRepeats)` which returns a new sound that is the same as the given sound except that the portion of the sound of size length starting at the offset is now repeated numRepeats times to achieve a stuttering effect. Criteria For Success: Test your function on a sound like always.wav. Select any reasonable start and end positions and a repeat values around 5 to 10. Explore and play the resulting sound and you should observe the stuttering effect.

You can achieve this by using the `copyChunk` method as shown below:

```
def stutter(self,offset,length,numRepeats):
    result = Sound(self.getLength()+(numRepeats-1)*length)
    # copy before the stutter
    self.copyChunk(_____________________)
    # stutter numRepeats times
    for i in range(0,numRepeats):
        self.copyChunk(___________________________)
    # copy the portion after the stutter
    self.copyChunk(____________________________________)
    return result
```

Criteria of Success: Test your method on a sound like always.wav. I used an offset of 5000 and a length of 100 and stuttered with at least 5 repeats.

Submit your python files with your methods in Canvas for grading.