CS 100 Programming I
Project 1

Revision Date: October 2, 2015

Preamble

You may develop your code anywhere, but you must ensure it runs correctly on a Linux distribution before submission.

Music!

> "He who hears *music*, feels his solitude peopled at once" - Robert Browning

Your task is to develop a filter to remove the initial white noise in an audio file.

You will need to make your own audio files for testing. It is permissible to share your audio files (but not your code).

Getting the necessary bits

Download the audio player *rplay*:

```
wget http://troll.cs.ua.edu/cs100/python/projects/rplay
```

Download the audio file converter:

```
wget http://troll.cs.ua.edu/cs100/python/projects/rra2wav.py
```

Download an audio file for testing:

```
wget http://troll.cs.ua.edu/cs100/python/projects/mandolin.rra
```

Install Sox, an open-source audio processor:

```
sudo apt-get install sox
```

Make sure Python3 is installed (it should already be installed).

You should now be able to hear the song play with the command:

```
rplay mandolin.rra
```

Input/Output

The threshold value and the song to be stripped of its initial "silence" are to be supplied as command line arguments, as follows:

```
python3 clean.py THRESHOLD_VALUE ATTACK_TIME RRA_FILENAME
```

The `THRESHOLD_VALUE` command-line argument is an integer value which is used to remove initial data values whose absolute values are below the given threshold. The reason for the threshold is that silence is rarely the absence of sound, but rather a very low sound level. Note that the term 'absolute value' indicates that any value between the *positive* or *negative* threshold value is ignored during the initial audio data scanning. The `ATTACK_TIME` argument specifies how much of the initial noise should be left so that the audio doesn't start abruptly, in terms of milliseconds. The `RRA_FILENAME` specifies the file name of the audio file you wish to process. Here is an example call to the program.

```
python3 clean.py 20 100 song.rra
```

This strips out all the initial data until 100 milliseconds before a value greater than 20 or less than -20 is encountered. The output of the *clean.py* program is sent to the console. This next command sends the output directly to the *rplay* utility, which plays the song:

```
python3 clean.py 20 100 song.rra | rplay
```

Instead of playing the output, you can save it to a file for later playback:

```
python3 clean.py 0 150 song.rra > new.rra
rplay new.rra
```

This saves the processed audio in a file named *new.rra*. You can look at both the original file and the newly saved file using *vim* and you can play the new file using *rplay*.

RRA file

RRA stands for *readily readable audio*. It is designed to be easily viewed (and debugged) and to be extensible. Originally suggested by a student named Ian Taylor for use in a High School Programming Contest, it has been adopted by Dr. J. C. Lusth in his music research. The RRA file is composed of two sections, a header section and a data section. The header of the RRA file is as follows:

```
RRAUDIO
...
%%
```

The header must start with the token `RRAUDIO`, while the token `%%` indicates the end of the header. The RRA sound data follows the header as shown below:

```
RRAUDIO
%%
0
0
1
-2
...
```

In between the `RRAUDIO` token and the `%%` token, information about the audio file may appear:

```
RRAUDIO
sampleRate: 44100
samples: 2345102
channels: 1
createdBy: wav2rra
%%
0
0
1
-2
...
```

The sample rate has the units of samples per second. This value will be useful in figuring out how many samples to save before the threshold is reached.

Scanner class usage

To easy the reading of free-format data files (such as RRA files), a python module has been prepared for you. You can download the module using the following linux command.

```
wget http://troll.cs.ua.edu/cs100/python/projects/scanner.py
```

The scanner is a module which makes the reading of a structured input file quite easy. CS250 and CS260) and is very easy to use. The following is the *example-code* to read an RRA header:

```
from scanner import Scanner
def process(fileName):
    s = Scanner(fileName) #opens the file for reading

    token = s.readtoken() #should be RRAUDIO

    # read the first attribute
    attribute = s.readtoken()
    while (token != "%%"):
        value = s.readtoken()
        # process the attribute-value pair here
        print("found attr-value pair:",attribute,value)
        # read the next attribute
        attribute = s.readtoken()

    # always close the files when finished!!
    s.close()

process("song.rra")
```

Note that a token is defined as any contiguous span of characters that is not whitespace (e.g. space, tab, newline).

1 Processing command-line arguments

Here is a short python program that checks if there are three command-line arguments and, if so, prints them out:

```
import sys

def main():
     # check to see if there are three arguments
     # look at the length of sys.argv
     # sys.argv holds the name of the program plus additional arguments

     if len(sys.argv) != 3:
        print("incorrect number of arguments, should be 3 (including program name)")
        sys.exit(1)

     print("the name of the program is",sys.argv[0])
     print("argument 1 is", sys.argv[1])
     print("argument 2 is", sys.argv[2])

main()
```

All command-line arguments are strings, so you will need to convert the threshold value and attack time arguments to integers. That is to say, you must know how to convert a string of digits:

```
"150"
```

into a number:

```
150
```

To make the conversion, you can use an expression like this:

```
anInteger = int(aStringOfDigits)
```

where *anInteger* is a variable to hold the resulting integer and *aStringOfDigits* is a variable that holds the string that looks like a number.

Handling the attack

You need to save the last n amplitudes when reading the RRA data, where n is determined from the sample rate and the attack duration. That way, when you detect a value above the threshold value, you can print out the saved data before printing the remaining data.

A clever way to do this is with a *circular buffer*, which is simply an array. In this case, the circular buffer will have size n. When starting, the buffer is empty and the index i, which marks the first available slot in the buffer, is set to zero. As you read in values, you store them in the buffer at ever higher indices. After the buffer becomes full for the first time, storing a new value in the buffer always replaces the oldest value in the buffer. In general, the oldest value in the buffer is at index zero if the buffer is not full and at index $(i + 1)\%n$ if it is.

You can read more about circular buffers on the internet. Make sure you handle the cases where there is no or very little silence/noise at the beginning of the audio file. That is, print out what data has been saved, even if the buffer is not full.

Compliance Instructions

First download a copy of the *test0.rra* file:

```
wget http://troll.cs.ua.edu/cs100/python/projects/test1.rra
```

Then test your program with the following command:

```
python3 clean.py 0 0 test0.rra
```

The output should be:

```
RRAUDIO
samples: 10
channels: 2
sampleRate: 44100
bitsPerSample: 16
createdBy: mrrao
%%
35
48
132
52
29
11
134
467
1210
21034
```

A program that fails with a syntax or runtime error on this test file will be given a score of zero.

Submission Instructions

Change to the directory containing your assignment. Do an *ls* command. You should see something like this:

```
clean.py test0.rra scanner.py wav2rra.py rplay mandolin.rra
```

as well as your audio files for testing.

You can now submit using the following command:

```
submit cs100 xxxx project1
```

Replace `xxxx` with your instructor's **nixie** login name.