CPS109 Lab 9

Most of the questions in this lab come from Chapter 7 of the course text, Introduction to Computing and Programming in Python, by Guzdial and Ericson. Please put your answers (numbered) in a document and submit it on D2L as a PDF file. Other formats are not accepted.

The **learning objectives** for Chapter 7 include:

- to understand digitization of sound and limits of human hearing
- to use the Nyquist theorem to determine sampling rate
- to manipulate volume
- to create and avoid clipping
- to use arrays as a data structure
- to use the formula that n bits allows for 2^n patterns
- to use sound objects
- to understand scope of a variable

To do:

For each question, include in your solution document your code and your example picture.

1) Define briefly in your own words the following terms: **clipping, normalize, amplitude, frequency, rarefactions**.

2) Write the following numbers in two's complement: -9, 4, -27. Hint: start with the binary version of the positive number. If the number is positive, you are done. Otherwise complement the bits and add 1 to the result. Complementing means changing 0 to 1 and vice versa.

3) Write a function **twoscomplement**(**binarystring**) which works on an input binary string, which we assume represents the positive number that we want to make negative, by first flipping all the binary digits (complementing), and then "adding 1". You can assume that the input string represents 16 bits. For example twoscomplement('0000000000000110') is simulating generating the two's complement of -6, since the input value would represent 6 if it were a binary number rather than the string it is. Doing the complement you would get '1111111111111001', and then adding 1, you would get '1111111111111010', since we are adding 1 to the rightmost bit, which generates 0 and carry 1. Note: you must use string operations and loops, not some special builtin python library function.
4) Write a function intToBinaryString(n) which takes a positive integer n and returns a string representing it's binary value in 16 bits. For example, intToBinary(4) should return '0000000000000100', and intToBinary(27) should return '0000000000011011'. To obtain the binary string, to something like the following, which I illustrate with n = 27:

- 27 % 2 = 1 so 1 is the rightmost bit, and divide 27 by 2 to get 13
- 13 % 2 = 1 so the next bit on the left is 1, and divide 13 by 2 to get 6
- 6 % 2 = 0 so that's the next bit, and 6/2 = 3
- 3 % 2 = 1 the next bit, and 3/2 = 1
- 1 % 2 = 1 the next bit and 1/2 = 0,
- 0 % 0 = 0 that's the next bit, and 0/2 = 0, and so on for the rest of the bits
- Read the bits in the up direction: 0000000000011011

Note: you must use string operations and loops, not some special builtin python library function.

5) Write a function **doubleAmplitude(soundfile)** which takes in a WAV file, like croak.wav, converts it to a sound object, then get the samples, double the values and play the louder sound.
6) Write a function **doublehalf(sound)** to double the volume (amplitude) for all the positive values and halve the amplitude for all the negative values. Try it on 'test.wav'. Can you still understand the words? Copy your function to **zeronegative(sound)** which does not change

the positive sample values, but changes negative values to zero. Try it on test. Can you still understand the words? Explore(test) to verify that you did what you think you did.

7) Write a function **minValue(sound)** to find the smallest value (probably negative) in a sound and print it out. Try it on test.wav and say what the value is.
8) Write a function **countZeros**(**sound**) which counts and returns the number of times the sameple is 0.
9) Write a function **clip**(**sound**, **maxvalue**), which changes any sample value that is more than maxvalue to maxvalue, and any sample value that is less than -maxvalue to -maxvalue. Try it on the test sound with a maxvalue of 1000. Is the sentence understandable still? Check the appearance of the wave with explore(test).
10) Write a function **falseIncreaseVolume**(**sound**, **increment**), which adds increment to each sample value of the sound. This is the wrong way to increase the amplitude. Try it on test.wav with an increase of 5000. How does the sound change?