Name:

|  |  |  |
| --- | --- | --- |
| Problem | Points | Score |
| 0 | 50 | Simply show up… |
| 1 | 60 |  |
| 2 | 70 |  |
| 3 | 80 |  |
| 4 | 90 |  |
| 5 | 100 |  |
| Total | 100 |  |

Notes:

1. For this exam you are allowed to open a terminal window on your computer, you are allowed to web surf with Google, but you cannot use online chat or other interactive services.
2. Your code and results should be placed in directories p01, p02, …, p06.
3. You must work the problems in order – you cannot skip a problem. For example, you cannot complete p06 until you have finished p05.

As usual, place your work here with the proper permissions:

/data/courses/ece\_1111/current/exams/exam\_04/lastname\_firstname

You must use Python 3 for this exam (the version of Python on the AWS server).

This exam is structured in a tiered manner. Start with problem 1. When done, copy your code to the next problem and continue editing it. Only turn in code that is completely working. I will grade the highest level you submit. If it doesn’t work, you will get a 50 for this exam (we call this the “you didn’t debug your code and wasted my time” penalty). Therefore, what you submit must work and meet the stated requirements.

If you write good clean code for problem no. 1, you can reuse that code for the remaining problems and quickly solve those problems.

**Problem No. 1**: Write a program that generates a binary file (16-bit signed integers) containing a sinewave. The interface should be this:

p01.py <amplitude> <frequency> <phase> <sample\_frequency\_in\_Hz> <duration\_in\_secs> <filename>

For example, this command line:

p01.py 10000 100 45 1000 10 p01.raw

produces a sinewave that follows this equation:

$$f\left(n\right)=10000\*sin\left(2π100\*\left({n}/{1000}\right)+\left({45}/{360}\*2π\right)\right)$$

and writes the sample values to a binary file containing 16-bit integers. Use the “od -s” command to verify the contents of your file are correct.

Implement this by creating a class named GenSinewave that includes a function named “compute” that returns a sample value given the parameters of the sinewave. Add a second function, “generate”, that takes an initial time and a duration, and generates a vector of sample values. Create a third function, called “create” that takes a filename as an argument, and creates a file with the appropriate data.

The arguments of the sinewave should be passed via a constructor or through a set method. They should not be arguments to “create”.

**Problem No. 2:** Write a program that generates a noise signal by writing it to a binary file. The interface should be:

p02.py gain <sample\_frequency\_in\_Hz> <duration\_in\_secs> <filename>

The argument “gain” controls the amplitude of the noise, and is limited to the range [0,32767]. A value of 0 should produce a file of zero values. A value of 32767 should produce a signal that falls in the range [‑32767, 32767]. Use a uniform random number generator for this task that outputs values in the range [­‑1,1], and scale its output to the range [‑gain, gain].

Implement this by creating a class GenNoise that has a method “compute” that generates the noise data.

**Problem No. 3:** Write a program that computes the linear combination of two signals by adding then together:

p03.py <input\_filename\_1> <input\_filename\_2> gain <output\_filename>

This command creates an output file that contains sample values that are a weighted sum of the input signals:

sig\_out[n] = sig1\_in[n] + gain \* sig2\_in[n]

Check the result with “od -s”.

**Problem No. 4:** Write a program, p04.py, that computes the mean and standard deviation of a signal:

p04.py <filename>

the mean = xxxxx.xxx

the standard deviation = xxxxx.xxxx

**Problem No. 5:** Write a program, p05.py, that computes the mean for a “window” of a signal. The interface should be:

p05.py <filename> <sample\_frequency> <center\_time> <duration>

This program converts center\_time to the correct sample value, and converts duration to a number of samples. The program seeks into the file and reads the proper number of samples so that the middle sample is from the sample corresponding to center\_time. We call this a “window” of data centered around center\_time.

The program should compute the mean of this data and output it.