Name:

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

Notes:

1. This exam must be completed in Python 3. Place your solution in:

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

1. You are allowed to web surf with Google and use other tools, like ChatGPT, but the code you turn in must be your own and must demonstrate mastery of the material. Simply copying ChatGPT code, or using code that you do not understand, is unacceptable.
2. Your code and results should be placed in directories p01, p02, …, p05. Create one MS Word document, MySolutions.docx, and place in the root directory of your workspace (/data/courses/ece\_1111/current/exams/exam\_04/lastname\_firstname). Put all your explanations in this one document. Use a heading “p01” for the first solution, “p02” for the second, etc. Start a new page for each solution.
3. You must work the problems in order – you cannot skip a problem. For example, you cannot complete p05 until you have finished p04.

This exam is structured in a tiered manner. Start with problem no. 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 that level doesn’t pass my test cases, 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 (-bit signed integers) containing a sinewave. The interface should be this:

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

For example, this command line:

produces a sinewave that follows this equation:

and writes the sample values to a binary file containing -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 in seconds and a duration in seconds 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”.

Run your program using this test case:

In your MS Word document, show the output of your program along with the output of “od -s” and explain why your output is correct.

**Problem No. 2:** Write a program that sum two files together sample by sample:

This program loops over all samples in each file and computes the weighted sum:

Demonstrate this works by running this command:

*myprog.py 1.0 <file1> -1.0 <file1> <output\_file>*

and showing that the output file contains all zeroes using “od -s”. If it does not generate all zeroes due to rounding errors, fix it! Demonstrate your output for this test case in the MS Word document.

**Problem No. 3:** Write a program that computes the histogram of the magnitude of the amplitude of a signal. Use a bin size of . Your first bin should extend from , the second bin from , etc. Generate a sinewave with an amplitude of 32,767 using your code from No. 1:

*../p01/myprog.py 32767 1 0 100 10 sine.raw*

and write the resulting histogram to stdout, showing the range of the bin and the number of samples that fell into that bin. Copy and paste the terminal output into an MS Word document and explain if your results make sense.

**Problem No. 4:** Write a program that computes the average value from your histogram and the average value of the signal, and compares them:

*p04.py <histogram.txt> <signal.raw>*

In your MS Word document, discuss why these numbers are similar or different depending on your findings.

**Problem No. 5:** Write a program that reads an matrix from a file in this format:

3, 3

1,0,0

0,1,0

0,0,1

and multiplies every three samples of a signal stored in a -bit binary file by this matrix, producing a new vector. Write that vector to an output file:

*myprog.py <matrix file> <signal.raw> <output\_file.raw>*

You can use whatever matrix/vector multiplication tool you want. Read the signal file into memory, loop over it samples at a time, and process it. Your output file should be exactly the same number of samples as your input file. If the length of the file is not a multiple of , you can assume values of zero for the missing data.

Verify your program is doing the right thing using the above matrix. It should produce an output file identical to the input file. The Unix command “diff” should produce no output.

Then use this matrix:

6, 6

2,0,0,0,0,0

0,2,0,0,0,0

0,0,2,0,0,0

0,0,0,2,0,0

0,0,0,0,2,0

0,0.0.0,0,2

to further verify your program. Document your verification of your code in your MS Word document.