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, …, p05.
3. You must work the problems in order – you cannot skip a problem. For example, you cannot complete p05 until you have finished p04.

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 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.

Also, please resist the temptation to use code generation tools such as ChaptGPT. It is easy to spot code generated from these tools, and the solutions these tools produce often aren’t within the scope of this introductory class.

All of the problems in this exam are going to use this dataset:

ece-000\_[1]: p

/data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph

ece-000\_[1]: find . -name "\*.txt" | sort -r

./f\_d0\_00.txt

./d1/f\_d1\_cc.txt

./d1/f\_d1\_bb.txt

./d1/f\_d1\_aa.txt

./d1/f\_d1\_01.txt

./d1/f\_d1\_00.txt

./d1/d2/f\_d2\_99.txt

./d1/d2/f\_d2\_02.txt

./d1/d2/f\_d2\_01.txt

./d1/d2/f\_d2\_00.txt

./d1/d2/d3/f\_d3\_02.txt

./d1/d2/d3/f\_d3\_01.txt

./d1/d2/d3/f\_d3\_00.txt

Do not copy this data into your exam workspace. Work with this data “in place” meaning that your program should accept any pathname (e.g., “/data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph”) as input.

Note that while most of the filenames end in “NN.txt”, where “NN” is a two-digit sequence, some end in “LL.txt” where “LL” is a two-letter sequence.

**Problem No. 1**: Write a Python program that recurses through a directory tree, finds all files ending in “NN.txt”, where “NN” is a number, sorts them in ascending order, and prints them in this format:

ece-000\_[1]: p01.py /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph

00: /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph/f\_d0\_00.txt

01: /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph/d1/f\_d1\_01.txt

02: /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph/d1/f\_d1\_00.txt

03: /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph/d1/d2/f\_d2\_99.txt

04: /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph/d1/d2/f\_d2\_02.txt

05: /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph/d1/d2/f\_d2\_01.txt

…

This is essentially the result of a “find <dir> | sort -r” command with a bit of filtering. Your program should read the files into a list, sort the list and remove any files that don’t meet the required naming convention (“\*NN.txt”).

**Problem No. 2:** Using your code from problem no. 1, write a program that reads the data in each file into a vector. For each line in the file, the first entry is the index of the value in the vector, and the second entry is the value. For example:

ece-000\_[1]: more f\_d0\_00.txt

0 0.0

3 9.0

1 1.0

4 16.0

5 25.0

2 4.0

6 36.0

7 49.0

9 81.0

8 64.0

10 100.0

The second line evaluates to v[3] = 9.0.

Compute the average value of this vector over all the files read, and print it out:

ece-000\_[1]: p02.py /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph

average:

 v[0] = 0.0

 v[1] = 1.0

 …

Since all the test files in my test data are the same, the average should be identical to the values in the first file.

**Problem No. 3:** Write a program (p03) that computes a matrix of dot products for all the vectors found in the path specified. Print the matrix using the standard Python print command. A dot product is defined as the sum of the element by element products:

$$dotproduct\left(x,y\right)=\sum\_{i=0}^{d-1}x\left[i\right]\*y[i]$$

where $d$ is the dimension of the vector.

Suppose there are a total of 10 files in your test data example. The matrix of dot products will be $10x10$. There is a Python function call you can use for a dot product.

**Problem No. 4:** Write a program, p04.py, that computes the minimum and maximum value of the matrix produced in problem no. 3:

ece-000\_[1]: p04.py /data/courses/ece\_1111/current/exams/ex\_04/picone\_joseph

min = 0.0

max = 100.0

Manually review the matrix values and make sure you are producing the correct values for the min and max. You might want to try a modified data set in which there are only a couple of files and they have different values.

**Problem No. 5:** Though we have not talked about eigenvalues and eigenvectors in this class, here is your chance to show that you can implement code even if you don’t understand the underlying math ☺

Rewrite the program in problem no. 3 to do an eigenvalue and eigenvector decomposition of the matrix. You can find many web pages showing you how to do this in Python (a one-liner). Print the eigenvalues and eigenvectors using any format you think is informative. Style points count ;) Note that the matrix for my test data is singular (non-invertible), so you will want to make a copy of the data set and change some of the values. You will study eigenvalues and eigenvectors in ENGR 2011.

To test your program, manually set your matrix to values shown in whatever tutorial you are using, This is a common debugging trick to isolate problems. In your driver program, before you call the eigenvalue solver, set the matrix to something simple for which you know the answer. Once you have verified your code works for these test cases, then you can remove this debugging code and let the function operate on your real data.