CME211 – Exam 1 – Python

October 24, 2019

Gradebook

The task of this programming assignment is straightforward: you are given as primary input a text file with student grades, and you are asked to compute their final course scores according to a syllabus that we will describe below. Your program can be either procedural or object-oriented, and we will provide implementation hints for either case.

Purpose The purpose of this exam is to test your Python software development skills. Besides Python coding, this includes algorithm design and complexity estimation, implementation design, code verification, and documentation.

Specifications All of your code should be written in a `gradebook.py` file. You are not allowed to use NumPy, SciPy, Pandas, or any other Python module that provides any matrix implementation which could also be used to represent a spreadsheet of grades.

Syllabus

The course you will be in charge of grading will follow a syllabus quite similar to what we have in 211. There will be m *homework assignments*, equallly splitting 50% of the total grade, and there will be exactly two exams which are evenly split amongst the remaining 50% of the grade. For *each* homework, students will be given numeric scores in the unit interval $[0, 1]$ describing their performance in a variety of k different *categories* e.g. we could think of these as things like *correctness*, *design*, and *writeup*. The number of homework assignments m, the number of grading categories k within each assignment, and the actual weights $w_1, \ldots, w_k$ for each category will be given to you as inputs in addition to a raw grades file. In order to compute the final score associated with each homework, you must compute a weighted sum where you grant the scores for each category are multiplied by their corresponding weight. The lowest homework score shall be dropped before computing the homework portion of each student's grade; in computing the average homework score we will weight all final homework assignment scores *equally*.

Program Interface Your program should be runnable if provided an input file specifying raw student grades, the number of homework assignments m, and the number of scored *categories* per assignment k; i.e. the interface to run your program should be as follows:

```
$ python3 gradebook.py grades_file output_file m k [w_1 w_2 ... w_k]
```

Here, `grades_file` specifies the input file describing raw student grades, `output_file` specifies the file we wish for you to write your program's output to, $m > 1$ is a positive integer describing the number of homework assignments (i.e. there are always at least two assignments such that you may drop the lowest and still have at least one remain), $k > 0$ is a positive integer describing the number of categories scored for each assignment, and $w_1, \ldots, w_k$ are real numbers specifying the *weights* for each of the k categories. If w_i's are not specified by the user of the program, they should default to each taking on the value $1/k$ i.e. we place equal weight on each category unless specified otherwise. Your task is to create a program that ingests the raw grades and writes the listing of final scores for each student-id in the input to an output file.

Input File Format

The input file will contain a single row per student in the course roster. The first element on the line will be the student id, which can contain arbitrary characters, can vary in length, and could start off with leading zeros which are meaningful. The $m \times k$ homework scores will *always* follow where the convention is that we list homeworks in ascending order (`hw_0`, `hw_1`, ..., `hw_m-1`) and the k scores for each assignment will always be ordered consistently across assignments and across students. The two exam scores will always be the last items to appear on each line. We can assume input files are well formed, i.e. that there are *always* $m \times k + 3$ elements on each line where all but the first element are scores lying in the unit interval. Although the input files will *not* contain a header describing the columns, you could envision a header that describes our ordering as follows:

```
student_id hwk_0_cat_1 hwk_0_cat_2 ... hwk_0_cat_k hwk_1_cat_1 ... hwk_m-1_cat_k exam_1 exam_2
```

So, suppose that $m = k = 3$, e.g. that there are three assignments with three categories associated with each. Then an input file describing two students' grades could look as follows:

```
15730258 0.8 1 0.95 0.9 0.95 1 0.75 0.825 1 0.765 0.815
0298452 0.75 0.925 1 0.95 0.84 0.9 1 1 1 0.86 0.88
```

Notice that we are use *spaces* to delimit elements on each row of our input file. The number of students in a class can be arbitrary, and we emphasize that there can be an arbitrary number of assignments, i.e. in particular there can be more or less than three assignments as depicted above, and there can be more or less than three categories per assignment.

Output Format

We'd like an ordered listing of student grades, sorted from high to low. Each line of the output file should contain three elements: the student id, the (zero-based) index corresponding to the lowest scoring homework assignment that was dropped, and their final score which lies in the unit interval $[0, 1]$ (rounded to three decimal places). In the event that multiple homework assignments are tied for having the lowest score, arbitrarily choose one to drop. Just like the input file, we'd like for the output file to be space delimited. An example output that does *not* correspond to the example input provided above, but which does demonstrate the desired format could be:

```
01253 0 0.950
91285 6 0.900
00248 3 0.765
```

E.g. the student with the highest score (0.95) dropped their first assignment, and the student with the next highest score (0.90) dropped their seventh assignment.

Exam Questions (**Functionality 65 points, Design 25 points, Writeup 10 points**)

The points below are requirements for your program. They are included to help you organize the process of building a working program, and are expected to be included in your submission.

Sanitizing inputs

You can assume that m and k will be sanitized inputs and strictly positive integers, i.e. that $m \geq 2$ and $k \geq 1$. However, you'll want to verify yourself that if weights $w_1, \ldots, w_k$ are provided, that they each lie within the unit interval $[0, 1]$, *and* that they sum to one. If either of these conditions fails to hold, you should raise an exception with a helpful but generic message suggesting what the requirements on $w_1, \ldots, w_k$ are.

Function to compute assignment scores

You'll need to take the individual components $x_1, \ldots, x_k$ of assignment scores and aggregate them according to the weighted sum which places w_i weight on element x_i. Since k can vary across invocations of our program, we need to implement this function in a way that is *extensible*. You should ideally implement a function or method which, given a listing of raw homework scores outputs a computed homework score after applying the weights to each category.

Function for data ingestion

Implement a function which, given a string describing the name of a datafile, processes the file line by line and sets up the objects necessary for your program to continue executing. If you choose a procedural approach, this could mean return a tuple of variables, one which describes the homework listings and the other which describes exam scores; or, if you choose an object oriented approach it could involve returning a listing of `Student` objects. You may end up invoking your method which computes assignment scores as part of this sub-routine.

Determine lowest homework score for each student

We ask that you implement *either* a function or a method (depending on whether you choose a procedural or object oriented approach) which determines the lowest homework assignment score for each student. You'll then want to remove this lowest score before computing the assignment portion of their grade. In the event of a tie, you are free to choose arbitrarily one of the two candidates as the "lowest" scoring assignment.

Write sorted output

After sorting your students based on final course scores, you'll need to log the output to another textfile as specified by a command line argument. We ask that you use 3-digits of decimal precision when reporting final scores. The elements should be space separated.

Complexity of dropping lowest assignment score

Estimate the computational complexity of *your* function/method which determines the lowest homework score *and* removes it from the collection and write down your reasoning in your `README`. Be sure to take into consideration not just *finding* the lowest score but also *removing* it from the underlying collection. If you can think of a "faster" approach, please feel free to document your ideas in this short-answer portion.

Sample test input

It will be helpful as you debug your program to have a small sample input file which maps to a known output (which you can compute yourself by hand). One of the input files can test a trivial case, e.g. $m = 2, k = 1$ with just a few students, but we require you to write out two test files, call them `test_i.txt` for $i = 1, 2$ and to show how you might invoke your program on such inputs for some specified listing of weights in your `README` file. You should also explain what the known/expected output is for each case.

Writeup

We ask that you document your design decisions in a `README` file. In particular, you should justify why you choose to use either a procedural approach *or* an object oriented approach. If you do not have time to fully implement your solution, you should breifly explain how you would implement remaining sections of code. If you have fully implemented a solution, please comment on one area of code which could be improved if you had more time to work on the exam.

Program Requirements Summary

- Sanitize weights input.
- Implement a function to ingest data.
- Implement an extensible function or method to compute assignment scores.
- Implement an function or method to determine the lowest homework score for each student.
- Write output to file in specified format and order.

Writeup Requirements Summary

- Document design decisions, and justify your choice of approach.
- Explore either how you would improve your program, or how you would finish if you did not complete the exam.
- Discuss the complexity of your approach to finding and dropping the lowest grade.
- Discuss your expected output for your test cases and show how your program would be invoked on them.
- Include sample test inputs for $i = 1, 2$ in your submission (not explicitly as part of the writeup, but as separate files).

Implementation Hints

Procedural Implementation Should you choose a procedural implementation, we will leave the choice of data structures up to you. Eventually, you'll need to (i) determine the index of the lowest scoring homework assignment to drop for each student and (ii) calculate their final grade. After you've done all your computations, you might choose to store the student id, the index of the lowest scoring assignment, and the final grade all in a tuple. Since we'd like you to print the results in a sorted order, you can consider the following code to sort a `list`ing of tuples of length three based on the last element:

```
tuples = [("student1", 0, 0.95), ("student2", 3, 0.84), ("student3", 5, 0.975)]
tuples.sort(key = lambda x : x[2], reverse = True)
```

Object-Oriented Implementation It could also be natural to represent each student as an object with two key attributes: homework scores and exam scores. We might have a methods for determining the lowest ranking assignment and computing the final grades. Note that when it comes time to produce our ordered listing of student grades, we can choose to sort a list of students based on their final grade `score` as follows:

```
class Student:
  def __init__(self, ...):
    ...
  def ComputeGrade(self):
    ...
    self.score = ...
  def __lt__(self, other):
    return self.score < other.score
```

By overloading the *less-than* operator via the syntax `__lt__`, we've defined how two students can be compared on the basis of their final grade `score` attribute and in turn calling `list.sort(reverse = True)` on a `list` of `Students` will work nicely.

Submission Instructions

Within your Github repository, create a directory `exam1`, and add your submission files to it. Your submission should consist of

- `gradebook.py` file with your Python code solutions.
- `README.md` file with your writeup and documentation.
- `test_i.txt` files (for $i = 1, 2$) with small test cases that you create.

Commit your submission and push it to GitHub. Once when you are satisfied with your submission, tag it "release".

Tags in Github To tag your submission, first make sure that you committed and pushed everything to your GitHub repository. Then, set the tag to the local head by executing

```
git tag release
```

in your local git repository. To push the tag to GitHub, execute

```
git push origin release
```

Using the GitHub GUI to tag a release Note that it's also possible to use the Github web-interface to tag a release: simply navigate to your code on Github, ensure you have selected the `code` tab, then click "release" in the same pane which displays the number of commits and branches. The subsequent page has an icon to "Draft a new release".

Late submissions The time of your submission will be the time of the tagged commit. This time must be less than 3 hours from the time you first accessed the exam from Canvas. We will check out tag "release" from your repository and grade what is there, subject to the submission time being less than or equal to 3 hours from the time-accessed timestamp on Canvas. For exams that roll in after the 3 hours mark, will at first allow a 10 minute grace period, beyond that each additional minute spent on the exam will cost 10 points.