Lab One: Grade Calculator
Evan Conley, Matthew Bolenbaugh, Kyle Calvert
April 09, 2018

1. Problem Statement

Calculate a weighted grade given a list of homework, quiz, and test grades.

We are limited to using only the functions given out in our lab.

Requirements:

- main() - initialize data, call other functions
    - homework grades list
    - quiz grades list
    - test list
    - category weights
- Functions: Must utilize the functions as described in Step 2 of the lab instructions.
    - average(score_list, max_list)
    - letter_grade(percent)
    - average_weighted(score_list, max_list, weight)

2. Planning

Input grades of homework(s), quiz(s), and test(s). Output a finalized weighted grade given the input grades. Input will be in the form of separated lists describing the total points available and the actual grade for each assignment.

Steps to a weighted grade:

1. Take in the grades of one category (homework, quiz, and test)
2. Grade each category.
3. Grade the weighted grade given all grade categories.

Specifications for associated functions:

- average(score_list, max_list):
    - score_list: List of floats representing the scores received on specific assignments in given category.
    - max_list: List of floats representing the maximum attainable score on the respective assignments.
    - Returns a floating point number representative of the average grade for the category.
- letter_grade(percentage)

    - percentage: A floating point number between 0 and 1 representing a percentage score.
    - Returns a string from the set {“A”,”B”,”C”,”D”,”F”} representing the letter grade associated with the percentage score.
- average_weighted(score_list, max_list, weight)
    - score_list: List of floats representing the scores received on specific assignments in given category.
    - max_list: List of floats representing the maximum attainable score on the respective assignments.
    - weight: Represents the weighted average as a floating point number between 0 and 1 inclusive
    - Returns the weighted average as a floating point number.

3. Implementation and Testing

Due to the relatively simple nature of the lab, implementation was pretty straightforward. We utilized the Atom text editor with the installed Teletype package in order to share a workspace, coding together in real-time. After identifying our constraints and requirements, we began to review our attempted pseudo-code solutions and picked out strong points of each one.

As to the actual implementation, we started with naming and initializing the variables and lists needed for the lab. We then split up the definitions of the required functions. The average function was computed by summing up the score list and dividing it by the sum of the possible points list. Then, average weighted function called the average function on the score list and max points list, multiplying the result by a weight factor. Then, utilizing the constraint that the letter grade function took in a floating point value between 0 and 1, we multiplied the input by 100 and then tested against score ranges, returning String variables appropriate for each.

After getting the functions working, we all worked together to create and review comments, as well as to tinker with formatting options in the output until the desired effect of correct, readable output was achieved. Overall, we were a strong team, and the real-time collaboration tools made code review and work-split very easy to handle.

Running Image of Lab:

```
econmang@econDebian:~/Documents/CS/SchoolFiles/cs356/hw/labs/labone$ python GroupFiveKEMgrade_calculator.py
20% Homework Grade: 74%
20% Quiz Grade: 87%
60% Test Grade: 97%
Final Score: 91% (A)
```

4. Reflection and Refactoring

The solution our team created was fairly linear and utilized the list structure's built-in functions to quickly return summed values. By, also, utilizing the constraints on the lengths of the list of scores vs. max scores, we were able to divide the total summed points of a list by the summed total of max points to gain an average. Reviewing the structure of our program vs. that of the other pair programming group of our team, we saw different approaches in output found data. There was a general consensus that ours had a cleaner methodology of formatting the output to the display. A potential refactoring condition to implement if we were to complete this lab again could be to add in data validation checks so as to handle any extraneous inputs. This would increase the usability and safety of the program.