CSc 110, Spring 2017
Programming Assignment #5: Gradanator (20 points)
Due Tuesday, February 27th, 2018, 7:00 PM
Based on an assignment from the University of Washington by Marty Stepp

```
This program reads exam/homework scores
and reports your overall course grade.

Midterm 1:
Weight (0-100)? 10
Score earned? 78
Were scores shifted (1=yes, 2=no)? 2
Total points = 78 / 100
Weighted score = 7.8 / 10

Midterm 2:
Weight (0-100)? 10
Score earned? 84
Were scores shifted (1=yes, 2=no)? 2
Total points = 84 / 100
Weighted score = 8.4 / 10

Final:
Weight (0-100)? 30
Score earned? 95
Were scores shifted (1=yes, 2=no)? 1
Shift amount? 10
Total points = 100 / 100
Weighted score = 30.0 / 30

Homework:
Weight (0-100)? 40
Number of assignments? 3
Assignment 1 score? 14
Assignment 1 max? 15
Assignment 2 score? 17
Assignment 2 max? 20
Assignment 3 score? 19
Assignment 3 max? 25
How many sections did you attend? 5
Section points = 15 / 34
Total points = 65 / 94
Weighted score = 27.7 / 40

Quizzes:
Weight (0-100)? 5
Total points earned? 12
Total points possible? 12
Total points = 12 / 12
Weighted score = 5.0 / 5

Daily homework:
Weight (0-100)? 5
Total points earned? 5
Total points possible? 10
Total points = 5 / 10
Weighted score = 2.5 / 5

Overall percentage = 81.4
Your grade will be at least: B
<< your custom grade message here >>
```

This interactive program focuses on `if/else` statements, `input`, and returning values. Turn in a file named `gradanator.py`. The program reads as input a student's grades on homework, quizzes and three exams and uses them to compute the student's course grade.

Below is one example log of execution from the program. This program behaves differently depending on the user input; user input is bold and underlined below. Your output should match our examples exactly given the same input. (Be mindful of spacing, such as after input prompts and between output sections.) **Look at the other example logs on the course web site** to get more examples of the program's behavior.

The program begins with an introduction message that briefly explains the program. The program then reads scores in six categories: midterm 1, midterm 2, final, homework quizzes and daily problems. Each category is weighted: its points are scaled up to a fraction of the 100 percent grade for the course. As the program begins reading each category, it first prompts for the category's weight.

The user begins by entering scores earned on midterm 1. The program asks whether exam scores were shifted, interpreting an answer of 1 to mean “yes” and 2 to mean “no.” If there is a shift, the program prompts for the shift amount, and the shift is added to the user's midterm 1 score. Exam scores are capped at a max of 100; for example, if the user got 95 and there was a shift of 10, the score to use would be 100. The midterm's “weighted score” is printed, which is equal to the user's score multiplied by the exam's weight.

Next, the program prompts for data about midterm 2 and then about the final. The behavior for midterm 2 and the final is the same as the behavior for midterm 1.

Next, the user enters information about his/her homework, including the weight and how many assignments were given. For each assignment, the user enters a score and points possible. Use a **cumulative sum** as described in lecture.

Part of the homework score comes from sections attended. We will simplify the formula to assume that each section attended is worth 3 points, up to a maximum of 34 points.

The program then prompts for the information about the student’s quiz scores. The quiz score should be capped at the total points for quizzes the user entered.

The daily homework follows next. It is computed the same as the quizzes.

Once the program has read the user information for all categories, it prints the student's overall percentage earned in the course, which is the sum of the weighted scores from the six categories, as shown below:

```
Grade = WeightedMidterm1Score + WeightedMidterm2Score + WeightedFinalScore
      + WeightedHomeworkScore + WeightedQuizScore + WeightedDailyProblemsScore
```

$$Grade = \left(\frac{78}{100}\times 10\right) + \left(\frac{84}{100}\times 10\right) + \left(\frac{100}{100}\times 30\right) + \left(\frac{14+17+19+(5\times 3)}{15+20+25+34}\times 40\right) + \left(\frac{12}{12}\times 5\right) + \left(\frac{5}{10}\times 5\right)$$

$$Grade = 7.8 + 8.4 + 30.0 + 27.7 + 5 + 2.5$$

$$Grade = 81.4$$

The program prints a loose guarantee about a minimum grade the student will get in the course, based on the following scale. See the logs of execution on the course web site to see the expected output for each grade range.

90% and above: A; **89.99% - 80%:** B; **79.99% - 70%:** C; **69.99% - 60%:** D; **under 60%:** F

After printing the guaranteed minimum grade, print a custom message of your choice about the grade. This message should be different for each grade range shown above. It should be at least 1 line of any non-offensive text you like.

This program processes user input using `input`. You should handle the following three special cases of input:

- A student can receive extra credit on an *individual assignment*, but **the total points for homework are capped at the maximum possible**. For example, a student can receive a score of 22/20 on one homework assignment, but if their total homework score for all assignments is 63/60, this score should be capped at 60/60. Section points are capped at 34.
- **Cap exam scores at 100.** If the raw or shifted exam score exceeds 100, a score of 100 is used.
- **Cap quizzes and daily problems at 100%.** No extra credit should be possible in these two categories.

Otherwise, you may assume the user enters **valid input**. When prompted for a value, the user will enter an integer in the proper range. The user will enter a number of homework assignments ≥ 1, and the sum of the six weights will be exactly 100. The weight of each category will be a non-negative number. Exam shifts will be ≥ 0.

Development Strategy and Hints:

- Tackle parts of the program (midterm 1, midterm 2, homework, final exam, quizzes, daily homework) one at a time, rather than writing the entire program at once. Write a bit of code, get it to run, and test what you have so far. If you try to write large amounts of code without attempting to run it, you may encounter a large list of errors and/or bugs.
- To compute homework scores, you will need to cumulatively sum not only the total points the student has earned, but also the total points possible on all homework assignments.
- Many students get errors because they forget to pass / return a needed value, forget to store a returned value into a variable, and refer to a variable by the wrong name.
- All weighted scores and grades are printed with no more than 1 digit after the decimal point. Achieve this by calling the `round` function. The following code prints variable `x` rounded to the nearest tenth:

```
x = 1.2345
print("x rounded to the nearest tenth is", round(x, 1))  # 1.2
```

- Use `max` and `min` to constrain numbers to within a particular bound.

Style Guidelines:

For this assignment, you are limited to Python features from lectures 1 - 14. A major part of this assignment is demonstrating that you understand parameters and return values. Use functions, parameters, and returns for structure and to eliminate redundancy. For full credit, use **at least 5 non-trivial functions** other than `main`.

Like on previous assignments, you should not have print statements in your main function. Also, main should be a concise summary of the overall program; main should make calls to several of your other functions that implement the majority of the program's behavior. Your functions will need to make appropriate use of parameters and return values. Each function should perform a coherent task and should not do too large a share of the overall work. Avoid lengthy "chaining" of function calls, where each function calls the next, no values are returned, and control does not come back to main.

This document describes several numbers that are important to the overall program. For full credit, you should make at least one of such numbers into a constant so that the constant could be changed and your program would adapt.

Some of your code will use conditional execution with `if` and `if/else` statements. Part of your grade will come from using these statements properly. Review the portion of lecture 11 about nested `if/else` statements and factoring them.

Give meaningful names to functions and variables, and use proper indentation and whitespace. Follow Python's naming standards as specified in lecture. Localize variables when possible; declare them in the smallest scope needed. Include meaningful comment headers at the top of your program and at the start of each function. Limit line lengths to 80 chars.