Name _____

CPADS Exam #2 (Programming-redo)

Due: Sunday, 11-27-16 @ 11:59 pm

Download **CPADS_exam2_redo.py** and open it in **PyCharm**. Make the modifications that are indicated in the TODO section. You are free to reference your original **exam02** submission – you can download it from Marmoset if you don't have it. When you are done, submit **CPADS_exam2_redo.py** to Marmoset under the **exam02-redo** project. ***NOTE: YOU MUST DO YOUR OWN WORK - YOU MUST NOT DISCUSS THIS PROBLEM WITH OR RECEIVE ASSISTANCE FROM ANYONE ELSE.***

Problem: The grading policy for a certain computer science course is as follows:

1) There are a variable number of assignments. The assignment average is calculated from all of the assignments grades. Assignments can have up to 50% extra credit added to the grade.
2) There are 4 exams per semester. The lowest exam score is dropped, and the exam average consists of the three remaining exam scores. Exams can have a curve of up to 15 points.
3) The course grade is composed of one-third (1/3) of the average of the assignments, and two-thirds (2/3) of the average of the three highest exams.
4) In order to pass the course, the exam average must be at least 60%. If the exam average is less than 60%, the course grade is 0.0.
5) Course grades are assigned as shown in the table:

grade	% range
4.0	≥ 90
3.5	≥ 85 to < 90
3.0	≥ 80 to < 85
2.5	≥ 75 to < 80
2.0	≥ 70 to < 75
1.0	≥ 60 < 70
0.0	< 60

The skeleton code for a program that calculates the course grade for a student from their assignment grades and exam grades has been provided. Code to run the program, as well as run test cases against each of the three functions in the program has been provided.

NOTE: DO NOT MODIFY ANY CODE OTHER THAN THAT FOR THE FOLLOWING SIX TODO's.

TODO 1: Finish the code for the **getAssignmentGrades()** function. It should prompt the user for assignment grades until the user enters a negative value (grade < 0). The function should only accept valid ***integer*** grades (0 ≤ grade ≤ 150). After the user enters a negative value, the function should calculate the average of the ***valid*** assignment grades that were entered, and return that value. If no valid grades are entered, the function should return 0.0. Note that print and return statements have been provided at the end of the function.

TODO 2: Finish the code for the **getExamGrades(numExams)** function. The function has a single ***integer*** parameter, ***numExams***, which specifies the number of valid exam grades the user must enter. The function should only accept valid ***integer*** grades (0 ≤ grade ≤ 115). The function should continue to request exam grades until ***numExams*** valid grades have been entered. The function should then calculate the exam average after dropping the lowest grade, and return that average. Note that print and return statements have been provided at the end of the function.

Hint: You can drop the lowest grade by subtracting the minimum exam grade from the sum of the exam grades. Remember to divide by the correct number of exams after dropping the lowest grade.

TODO 3: Finish the code for the **getCourseGrade(assignAve, examAve)** function. The function has two ***floating point*** parameters: the assignment average, and the exam average. Calculate the course percentage from the sum of 1/3 the assignment average and 2/3 the exam average. Return the course grade using the above table, based on the course percentage, with the exception that if the exam average is less than 60.0, the assigned course grade should be 0.0. Note that print and return statements have been provided at the end of the function.

Name _____

TODO 4: Finish the code for the **getCourseName()** function so that it accepts the course name from the user. It should only accept 2 course names: CS101 and CS201 (with only that capitalization). The function should return the first valid course name the user enters. Allow the user a maximum of **MAX_ATTEMPTS** tries at entering a valid course name. If the user does not enter a valid course name within **MAX_ATTEMPTS** tries, return **INVALID_COURSE**.

TODO 5: Finish the code for the **getStudentID()** function so that it accepts the student ID from the user. It should accept only valid student ID's (**MIN_ID <= studentID <= MAX_ID).** The function should return the first valid **studentID** the user enters. The user can also enter a negative value to exit the function, which will be used by the **main()** routine to determine that the user is done entering grade data and is ready to get the course grade statistics and then exit the program. DO NOT enforce maximum attempts in this function

TODO 6: Finish the code for the **main()** function. There are three areas that you will have to complete:

1) Modify the **main()** function so that the user can enter grades for multiple students. You will have to select the lines of code to indent under the loop you are adding. The loop will run until the user enters a negative number for the student ID.
2) Add code inside of your loop to keep track of four statistics for the course:
    a. **courseSum:** The sum of all course grades (to be used later to calculate the average course grade).
    b. **numStudents:** The number of valid **studentID**s for which grades were entered.
    c. **courseMin:** The minimum course grade for the course
    d. **courseMax:** The maximum course grade for the course
3) Add code after the loop concludes that calculates the course grade average from **courseSum** and the **numStudents**.

CS100 Fall 2016

Name _____

NOTE: When you run the program, it will prompt whether you want to run the program ('***run***') or run test cases (enter anything else). The test cases for each TODO are given in the following table. If all of the test cases pass for a certain TODO, then there is a high probability that your code is performing correctly for that TODO.

1) Cells highlighted in green are terminating values for TODO 1.
2) Cells highlighted in blue contain values outside the boundary conditions for TODO's 1 and 2.
3) Cells highlighted in yellow contain values that will be dropped from the calculation in TODO 2.

Test Case	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Expected Value
TODO 1-1	72	-1						**72.0**
TODO 1-2	70	80	150	100	0	-10		**80.0**
TODO 1-3	60	70	151	90	95	101	-100	**83.2**
TODO 1-4	200	-1						**0.0**
TODO 1-5	-1							**0.0**
TODO 2-1	65	70	75	80				**75.0**
TODO 2-2	115	-1	116	60	80	72		**89.0**
TODO 2-3	70	70	60	38				**66.7**
TODO 2-4	1	0	85	86				**57.3**
TODO 3-1	150	115						**4.0**
TODO 3-2	80	95						**4.0**
TODO 3-3	89	90						**3.5**
TODO 3-4	87	84						**3.5**
TODO 3-5	84	85						**3.0**
TODO 3-6	78	81						**3.0**
TODO 3-7	79	80						**2.5**
TODO 3-8	72	77						**2.5**
TODO 3-9	74	75						**2.0**
TODO 3-10	72	69						**2.0**
TODO 3-11	70.5	69.5						**1.0**
TODO 3-12	59	60.6						**1.0**
TODO 3-13	59.9	60						**0.0**
TODO 3-14	1	89						**0.0**
TODO 3-15	150	59.9						**0.0**
TODO 4-1	CS101							**CS101**
TODO 4-2	CS201							**CS201**
TODO 4-3	cs101	Cs101	CS101					**CS101**
TODO 4-4	CS100	CS102	CS200					**invalid**

Name _____

Test Case	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Expected Value
TODO 5-1	903000000							**903000000**
TODO 5-2	903999999							**903999999**
TODO 5-3	902999999	904000000	903555555					**903555555**
TODO 5-4	0	999999999	1	-1				**-1**
TODO 5-5	-10							**-10**

TODO 6: For testing TODO 6, you will have to run the program, rather than run the test cases, and enter grades for at least two students. Use values from the test cases above for the assignment grades and the exam grades to come up with test cases values for TODO 6.

TODO 6-1: Verify that **main()** runs for multiple students, stops when a negative student ID is entered, and outputs the correct number of students.

TODO 6-2: Verify that **main()** calculates the course grade statistics correctly for one student and at least two students.

TODO 6-3: Verify that **main()** handles the situation where no students are entered.

Name _____

Python mathematical operators:

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	floating point division
`//`	integer division
`%`	modulo operator
`**`	exponentiation

Python logical operators:

`==`	equality
`>`	greater than
`>=`	greater than or equal to
<	less than
`<=`	less than or equal to
`and`	<logical expression 1> ***and*** <logical expression 2>, evaluates ***true*** if both are ***true***
`or`	<logical expression 1> ***or*** logical expression 2>, evaluates ***true*** if at least one is ***true***
`not`	**not** <logical expression>, evaluates to opposite value of <logical expression>

Outputting to the console:

```
print("character string", <integer>, <float>)
```

Inputting integer from the keyboard:

```
value = int(input("Enter an integer: "))
```

Inputting floating point number from the keyboard:

```
value = float(input("Enter a decimal number: "))
```

Using a *for* loop:

i increments, starting at 0, for ***value*** iterations. The body of the ***for*** loop contains all of the indented code.

```
for i in range(value):
      indented code
      indented code
```

Using a *while* loop:

Loops while <*logical expression*> is ***true***. The body of the ***while*** loop contains all of the indented code.

```
While <logical expression>:
      indented code
      indented code
```

Using if/elif/else:

If ***<logical expression 1>*** is ***true***, indented code under ***if*** executes, otherwise, if ***<logical expression 2>*** is ***true***, indented code under ***elif*** executes, otherwise, indented code under ***else*** executes.

```
if <logical expression 1>:
      indented code
elif <logical expression 2>:
      indented code
else:
      indented code
```