CIS 130 - Intro to Programming - Spring 2007
Homework Assignment #5 - **INDIVIDUAL** assignment

Homework #5 DUE**: BEGINNING** of class, Wednesday, March 7, 2007

Purpose: get practice with count-controlled loops

How to turn in: use the tool **~st10/130submit** on cs-server to turn in the files **hw5.py** and **hw5.txt** that you create below.

CONSIDER:

* Did you ever want to print something WITHOUT printing a newline at the end?

Here's an interesting-but-odd aspect of Python's print command: if you END a print command with a COMMA, it DOESN'T print a new line --- although it WILL print an extra blank!!

(Confession time: Python's print command adds an extra blank EVERY TIME you separate stuff with a comma!)

CONSIDER the following --- and PLAY WITH IT in the **python** interpreter until you believe it:

in file `play.py`

```
def playing():
    print "A" + "B" + "C"
    print "D", "E", "F"
    print "H" + "I" + "J",
    print "K", "L", "M",
    print "So long",
    print "Farewell!"
```

```
>>> from play import *
>>> playing()
ABC
D E F
HIJ K L M So long Farewell!
>>>
```

Create a file **hw5.py**, and create the functions described below. Paste evidence of testing them into **hw5.txt**.

1. SO: consider the above. You could use this in a simple count-controlled loop in a function **line_of_X**, that expects an integer as an argument and prints a line of THAT number of X's to the screen, where each X is followed by a blank (conveniently enough...!).

I'll even be nice and give you your Examples section:

```
# Examples: line_of_X(5) should cause this to be printed to
#           the screen:
# X X X X X
#           line_of_X(8) should cause this to be printed to
#           the screen:
# X X X X X X X X
```

PLEASE NOTE: Yes, Python has a feature that would allow this to be done without writing a loop. As the whole point of this problem is to practice writing a simple count-controlled loop, you MUST use an appropriate while-loop in a count-controlled fashion in your solution to receive full credit for your function.

2. Now, we'll be discussing writing **nested loops** - loops within other loops. But, many times, the easier way in these situations is to use auxiliary functions instead...! 8-)

   If you wanted to print something such as:

```
X X X
X X X
X X X
X X X
```

   ...then couldn't you do this by calling line_of_X(3) four times?

   And if you wanted to print something such as:

```
X X X X X X X X
X X X X X X X X
X X X X X X X X
```

   ...then couldn't you do this by calling line_of_X(8) three times?

   Well, then - write a function **box_of_X** that expects two integers as arguments, the number of rows in the desired "box" and the number of X's per row, and it prints a "box" of X's made up of that many rows and X's per row (again, conveniently with a blank after each X).

   That is, box_of_X(4, 3) should cause the following to be printed to the screen:

```
X X X
X X X
X X X
X X X
```

   ...and box_of_X(3, 8) should cause the following to be printed to the screen:

```
X X X X X X X X
X X X X X X X X
X X X X X X X X
```

   Oh, and your solution MUST use an appropriate while-loop in a count-controlled fashion and MUST call line_of_X appropriately to receive full credit.

3. But - you could do more than *that* with line_of_X. You could make a triangle, too:

```
X
X X
X X X
X X X X
X X X X X
```

For this simple count-controlled loop practice, write function **triangle** that expects a single integer as argument, and it prints a triangle with that many rows, with 1 X in the first row, X X in the 2nd row, X X X in the third row, ... until you reach the desired number of rows.

And, again, your solution MUST use an appropriate while-loop in a count-controlled fashion and MUST call line_of_X appropriately to receive full credit.

4. Someone in class commented a few weeks ago that the quiz_average --- the one that averaged five quiz grades - would be more useful if it could handle different numbers of grades.

We'll be able to pass such a set of grades as a single argument a little later -- but for now, we COULD use a count-controlled loop and interactive input.

Write **average_grades**, which takes one argument, the number of grades to be averaged, and interactively asks for that many grades. It totals/sums these grades as they are entered, and then computes and returns the average of those grades (being sure that a floating-point average is possible even if integer grades are entered).

You should, of course, use a count-controlled while loop in your answer.

5. Sometimes loops look to see if certain things are true along the way.

What if you wanted to know how many A-level grades there were in a collection of grades? (How many grades >= 90?)

You could read each in, and as you do so see if the latest grade entered meets the criterion for a grade of A. IF it is, you could increment an a_grade_count, or some specialized counter.

When the loop is done, you could say how many A grades had been entered by printing the value of that counter to the screen. You could even compute the percentage of grades that did so --- if 5 of 10 entered were A-level, you could say that ((1.0 * 5) / 10) * 100 = 50% of grades were A's.

Write a function **count_As** that takes one argument, the number of grades to be examined, and interactively asks for that many grades, counting the number of A-grades entered, and when done printing to the screen the number of A-level grades entered and the overall percentage of grades that were A's.

When you are happy with your files **hw5.py** and **hw5.txt**, type the following command at the cs-server prompt:

~st10/130submit

Then follow its directions to submit your files.