CS 1301 Extra Credit Problem Set
Due: Wed March 31st before 6pm. (NO Late Turn-In's accepted!)

Because this extra credit problem set is worth test points, you **MAY NOT look at other people's code!** You are allowed to talk with other students about how they solved the problems, and you may tell other students how to solve the problems, but you MAY NOT show other people your code or look at other people's code! If you talk with others you must include their names in your collaboration statement at the top of the file.

Each correctly solved problem is worth a test point. This extra credit problem set will allow you to gain up to 9 test points! This could raise your grade on an Exam up to 9%. However, grading is all-or-nothing, so make sure you test your code fully to make sure all your functions work!

Each problem will require you to write a function. Download the ec1.py file and write your functions at the top of the file, using the specified function name and parameter(s). When you want to test your functions, you can call the test() function that has been provided at the bottom of the file. The test() function will call each of your functions (if you have named them correctly!) and tell you if they pass a few simple tests. NOTE: If your function passes the included tests, it is likely to be correct, but TA's may find other problems when they read your code. Just because a function passes all the tests does not guarantee that you got it correct (but the likelihood is high!)

NOTE: Do NOT use global variables. Your functions must work using only local variables (those declared inside of the function body).

1. Write a function countJs() that accepts one parameter which it may assume is a string. This function should **return** an integer which is the count of how many "J"'s (capital letter j) it finds in the string.

2. Write a function maximum() that accepts one parameter which it may assume is a list. The list may have any elements upon which the < (less-than) operator is defined. Your function should **return** the "maximum" element of the list (as defined by the < operator. If the list is empty, your function should return None (of None-Type), which is easy to do by using a return statement with nothing behind it. ***YOU MAY NOT USE THE BUILT IN max() FUNCTION!***

3. Write a function changeGrade() that accepts a single parameter (a string). Your function should look for the following letters in the string, and **return** a new string with the appropriate substitutions made (note that you should only change capital letters!):

Replace:	With:
F	D
D	C
C	B
B	A

4. Write a function rotateText(textString, numToRotate) that accepts a parameter containing a string and "rotates" all letters (A-Z and a-z) by the specified number. By "rotate" we mean the following: If the numToRotate is 2, an "A" should be replaced by a "C", and a "B" should be replaced by a "D", and so on. If numToRotate is 5 an "a" would be replaced with an "e". After the string is rotated, the function should return the newly created string. Any character that is not a letter (such as numbers, whitespace or punctuation marks) should be unchanged.

   Note: Capital letters should be rotated into capital letters, and lower case letters should be rotated into lower case letters. Alphabets loop, so a "z" would convert to a "b" if numToRotate is 2. *Hints: The chr(num) function converts a number to a character. The ord('C') function converts a character to a number. The letters A-Z range from 65 to 90 and a-z range from 97 to 122. Interesting question: What happens if you apply the function twice, rotating 13 each time?*

5. Write a function blocksInPyramidHeight(height) that takes the integer height of a (2D) pyramid and returns the number of blocks in the Pyramid. Blocks in a Pyramid are layered over the join between two lower blocks, so a Pyramid of height 1 has 1 block, a Pyramid of height 2 has 3 blocks, a Pyramid of height 3 has 6 blocks, and so on. Hint: Each level has one more block than the last.

6. Write a function binaryToDecimal(binaryString) that takes one parameter made up of a STRING of zero's and ones (representing a positive/unsigned binary number). Your function should assume the parameter represents a decimal number, and **return** the corresponding decimal integer.

7. Write a function blocksInPyramidWidth(maxWidth) that takes one parameter representing the number of blocks at the base of the pyramid, and returns the number of blocks (total) in the (2D) pyramid. Pyramids are formed the same was as in problem 6. For example, a pyramid with 4 blocks at the base has a total of 10 blocks, while a pyramid with 2 blocks at the lowest level has a total of 3 blocks.

8. Write a function everyOtherLetter(aString) that returns a new string made up of every other letter in aString (starting with the first, skipping the second, including the third, and so on.)

9. Write a function sumNestedList(aList) that returns the sum of all numbers (ints and floats) in a list that may itself contain lists of numbers which may be nested to any depth. For example, sumNestedList([4,5, [2,3, [1.5,2]]]) should return 17.5. Any items that are not numbers should be ignored.

10. Write a function called tupleUnpacking(aTuple) that takes in a tuple of the following format: (aName, anInt). You can assume that aName is of type string, and that anInt is of type int. Your function should return a string formatted as follows (for any given aName and anInt): "aName likes the number anInt"

If you have any questions about how your functions should behave, look at the test code inside the test() function. This test code will show how the functions should be called and give default inputs and outputs.