CS 1301 Extra Credit Problem Set
Due: Wed October 1st 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 get help from others (or help others) you should include their names in your collaboration statement at the top of the file.

This extra credit problem set will allow you to gain up to 13 extra credit test points! This could raise your grade on Exam 1 by almost 10%. To receive full credit, you must complete all problems. Every problem you do not complete will subtract 4 point from the 13 possible points you receive. (Luckily, you can't earn negative points!)

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 problem 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!) Also, the output of some functions are not tested, or are left for you to test yourself.

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.

3. Write a function getFloat() that takes a string as a parameter. The function should print the string that is passed in as a parameter (using it as a prompt for the user) and then wait for user input. The function must attempt to convert the user input into a float and return it. If the user does not enter a valid float, the function should notify the user that they entered an invalid number, and ask them again, repeating until the user enters a valid number. Once the function successfully retrieves a valid float from the user, it should return it.

4. Write a function isPrime() that takes one parameter (you can assume this parameter will be in the form of a positive integer) and checks to see if it is prime. (A prime number is a positive integer that has exactly two positive integer factors, 1 and itself.) If the number is prime, your function should **return** True, if not, it should **return** False. [We will not test this function with any numbers larger than 10,000, so speed isn't terribly important.]

5. The pre-existing function getHumidity(zipCode) goes out on the internet and reads a weather website page for a specific zip code. (Note that the zipCode parameter expects a string!). Currently, it just prints the HTML from the webpage to the screen. Modify this function so that instead of printing the HTML to the screen, it **returns** just the current humidity at that zip code (as a float)!

6. 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:

If you find a:	Replace it with a:
F	D
D	C
C	B
B	A

7. Write three functions named countdownWhile(), countdownFor(), and countdownRecursive(). Each of the three functions should take one parameter, which they can assume is a positive integer. Each function should **print** a "countdown" starting at the number given and going down to 1, printing one number per line. After the functions reach 1, they should print "Done!" on the next line. In the first function, you may only use a while loop. In the 2nd function you may only use a for loop, and in the 3rd function you may only use an if statement and a recursive call. The output of all three functions are identical, but how they archive the output is slightly different.

8. Write a function decimalToBinary() that takes one parameter (you can assume this parameter will be in the form of an integer). Your function should assume the parameter represents a decimal number, and **return** a string made up of 0's and 1's representing the binary representation of that number.

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.