Lab 8 -- CPS109

This lab gives you more practice writing **functions**, including recursive functions, the rare use of global variables, and reading from a file.

There are seven programs for you to write. Only the functions are defined below, but your program should not only define the function, but also run it to show that it works. Put all of your programs into a .txt document and submit your single file on D2L

1. Recursion is where a function calls itself; (another example of recursion is where function A calls function B, and B calls C and C calls A, creating a loop in the calls). Some problems are most naturally solved recursively, such as writing the factorial function, or finding all the perumutations of a sequence, or checking if a string is a palindrome. Since those examples were done in class, here we will give you a toy example, which normally would not be solved recursively, but which illustrates the process. Write a recursive function to add a positive integer **b** to another number a, **add(a, b)**, where only the unit 1 can be added, For example **add(5, 9)** will return 14. The pseudocode is:
   # Base case: if b is 1, you can just return a + 1
   # General case: otherwise, return the sum of 1 and what is returned by adding a and b - 1.

2. Here is another toy recursion example, but with less guidance. Write a function **log2(x)**, which gives an integer approximation of the log base 2 of a positive number x, but it does so using recursion. Use a base case where you return 0 if x is 1 or less. In the general case, add 1 to the answer and divide x by 2. (This is purposely vague so you can figure it out yourself.)

3. Write a recursive function **reverse(sentence)** for reversing a sentence. For example, **reverse('Who let the dogs out?')** will return **'?tuo sgod eht tel ohW'**. The idea is to remove the first or last letter, reverse the shortened sentence, and then combine the two parts.

4. Write a recursive function **power(x, n)**, where n is 0 or a postive integer. For example, power(2, 10) will return 1024. Write a suitable base case, and for the general case use the idea that $x^n = x * x^{n-1}$.

5. A local variable in a function is either a parameter or a variable which appears on the left hand side (LHS) of an assignment statement in the function. A variable in a function is global if it is not local, but if you want to assign something to a global variable, **g**, in a function, then you will need the statement **global g**. Without the **global g** statement, assignment, like, **g = 5**, would make g local. You shouldn't use global variables very often when writing functions, since global variables reduce readability. Occasionally they are useful, such as when you would like to count how often a function is called. Define a global variable, **countcalls**, and increment it inside the power(x, n) function that you wrote for Q4, so that it counts the number of times the power function is called. Show that it produces the expected number of calls for power(2, 10) and power(5, 10) and power(5, 0), each separately..

6. Improve on your function from Q4, calling it powerHalf(x, n), where this function is recursive like power(x, n), but it also uses the idea that $x^n = (x^{n/2})^2$ when n is even. Use the countcalls variable (as in Q5) to verify that this version of the power function is more efficient.

7. Attached to this lab is a file representing a DNA sequence. The first line starts with '>' and is a comment, and the lines after that hold the sequence. The sequence has letters 'A', 'C', 'T', and 'G'. In your program for this question, read the sequence using the statements:

```
# ----------------------------------------------------q7
# Reading from a file
# ----------------------------------------------------
f = open('kdpF.txt')  # opens a file for reading
line = f.readline()   # reads a single line
print(line)
seq = ''
for line in f :       # reading the rest of the lines
   seq = seq + line
seq = seq.replace('\n', '')  # removing the newline characters
seq = seq.upper()

print(seq)

def gcContent(sequence) :
   #   You do the rest
   pass
```

Also write a function **gcContent(sequence)**, which returns the percent of the sequence which is either 'G' or 'C'. Use your function on the input sequence and print the results.