CS 215 – Activity 2.1 More Recursion

What is Recursion?

Recursion is simple – it is a function that calls itself.

True – but if that is all you think about, you will quickly learn about Stack Overflows (and I don't mean the helpful website).

You are right. I want to think about a base case and "smaller caller"

Right – the base case is a solution that everyone knows – almost like definition. The smaller caller – is a clever way to suggest that your *recursive step* moves closer to the base case.

Let me practice a few examples.

Practice

1. Write a program that reads values from a user until a zero is entered. Display the total of all the values entered. (write this program using both a loop and recursively)
2. Factorial and exponentiation are two operations that you usually implement early in your recursive experience.
   a) Factorial is defined as the product of all integers less than or equal to a given non-negative integer. It is express with an !
      3! = 3*2*1
      5! = 5*4*3*2*1
   b) Exponentiation is the mathematical operation where a number (the base) is multiplied by itself a certain number of times (the power). Of course you remember that, by definition, anything raised to the power of 0 is 1. (Work with positive, integer exponents for now)

   Create a recursive function for each of these.

3. We can use Newton's method to approximate a square root of number. The algorithm is as follows:

```
Get a number: x
Make a guess: x/2
While the guess is not good enough:
   Update the guess to be the average of guess and x/guess
```

   Write a recursive version of this approach. For the sake of this program, you should compute "good enough" to be when the difference between $guess^2$ and x is within 10^{-8}

4. The GCD of two positive numbers (n,m) is the largest number that divides evenly into both n and m. Euclid recognized that you could subtract the smaller of the two numbers from the larger and keep the same GCD. This leads to a nice recursive algorithm. See if you can develop it yourself, but keep in mind that nobody is expecting you to be the next Euclid. If your are stuck look for a description of the algorithm (make sure you understand it) and then implement it.

5. Write a recursive function that implements the `in` operator from Python. Given a list, the operator returns true when a target is in the list and false when it is not a member of the list. (Use an array in place of the Python list).

6. Consider a Singly Linked-List that is sorted. Imagine that you want to insert an element into the list and preserve the sort order. What would the recursive solution look like? Is it "better" than the iterative solution?