RECURSION

Definition:

Recursion means "defining a problem in terms of itself". This can be a very powerful tool in writing algorithms. Recursion comes directly from Mathematics, where there are many examples of expressions written in terms of themselves. For example, the Fibonacci sequence is defined as: F(i) = F(i-1) + F(i-2)

- recursion is a way of breaking up complex computational problems into simpler ones

- the same computation occurs repeatedly as the problem is solved, each time occurring on a simpler version of the problem

Simple Example:

For example, we can define the operation "find your way home" as:

1. If you are at home, stop moving.
2. Take one step toward home.
3. "find your way home".

Here the solution to finding your way home is two steps (three steps). First, we don't go home if we are already home. Secondly, we do a very simple action that makes our situation simpler to solve. Finally, we redo the entire algorithm.

Parts of a Recursive Algorithm

All recursive algorithms must have the following:

1. Base Case (i.e., when to stop)
2. Work toward Base Case
3. Recursive Call (i.e., call ourselves)

The "work toward base case" is where we make the problem simpler. The recursive call, is where we use the same algorithm to solve a simpler version of the problem. The base case is the solution to the "simplest" possible problem (For example, the base case to adding a list of numbers would be if the list had only one number... thus the answer is the number).

- Some problems can be solved both recursively and iteratively (with loops)
    - some problems are very difficult to solve without recursion, due to their more complex nature

Example of both recursive and non-recursive solutions:

Calculating a factorial value:

Non-recursive (using loops)

7! = 7 x 6 x 5 x 4 x 3 x 2 x 1

Recursive:

```
7! = 7 x 6!
    6! = 6 X 5!
        5! = 5 x 4!
            ...
                1! = 1        // base case - the end point
```

This problem meets the three requirements for a recursive solution:

1. the problem may be reduced to a series of repetitive elements

2. the problem has an identifiable endpoint (i.e. 1! = 1)

3. solving each of the smaller problems contributes to the solution of the bigger problem

Your Turn:

Try to determine a recursive solution to the following:

You want to compute the area of a triangle with a width of n, assuming that each square [] has area 1. (*Calculate the nth triangle number.*)

ex. []
[][]
[][][]

The third triangle number is 6.

ex. []
[][]
[][][]
[][][][]
[][][][][]

The fifth triangle number is 15.

Hint: What if the user enters 0 or a negative number? Is there more than one base case to stop the recursion?