SI@UCF Python Homework: Practice with Recursion

This assignment will consist of four relatively short required functions. For each part you must write a *recursive function* to help solve the problem. For fun, an optional part is included as well.

1) Write a recursive function that takes in a single non-negative integer and returns the sum of its digits. Here is the function prototype:

```
int sumDigits(n)
```

2) Write a recursive function that takes in a positive integer n, and prints out a triangle with n rows in the pattern shown below (for n = 3, spaces = 0):

```
  *
 ***
*****
```

Here's what should get printed out for n=3 and spaces = 2:

```
    *
   ***
  *****
```

Notice that not only must the number of stars on each row be accurate, but so must the number of spaces. n represents the number of total rows, and spaces represents the number of spaces starting the last row of the triangle structure. Here is the function prototype:

```
printTri(n, spaces)
```

3) A Generalized Fibonacci sequence is defined as follows, where a and b are real-valued constants:

$G_0 = a$, $G_1 = b$, $G_n = G_{n-1} + G_{n-2}$, for all integers $n > 1$.

Write a recursive function that computes the n^{th} term in a Generalized Fibonacci sequence given the values of a, b, and n. Here is the method prototype:

```
genfib(n, a, b)
```

Write each of these three methods in the file recday1.py. Then, write a main method in this file that allows you to adequately test all three recursive methods.

4) Write a function that computes the sum of a geometric sequence with first term `first`, common ratio `ratio`, with `n` terms. The function prototype is shown below:

```
geosum(first, ratio, n)
```

For example, a geometric sequence with first term 3, common ratio 2 with 5 terms is 3, 6, 12, 24, 48 and has a sum of 93.

5) (OPTIONAL - WRITE SEPARATE MAIN) This function will run in pyGame. Consider displaying a grid of 2^n x 2^n red squares, with a top left corner (x, y) and a side length of side. You may assume that the side length is a perfect power of 2. The squares should have a thickness of 1 pixel. The algorithm to solve this problem is as follows:

1) Draw the outer box.
2) Recursively call your function on four more grids representing the four quadrants of the outer box. Each of these four recursive calls should create grids of 2^{n-1} x 2^{n-1} red squares with the appropriate top left corners with half the side length of the original.

Here is the function prototype:

```
drawbox(n, x, y, side)
```

Here is the output for `drawbox(3, 50, 50, 512)`