Functions Calling Other Functions

Calling Functions

1. Predict the output of the following program. Run it to check your prediction.

```
def function1(x):
   return x**2
def function2(y):
   z = function1(y+1)-function1(y)
   return z
a = function2(3)
print(a)
```

2. Write two functions – *area_rectangle* and *area_triangle* – that calculate these values given appropriate values (e.g. length and width). *area_triangle* should call *area_rectangle*, and divide the returned value by two. Write a third function, *surface_area_pyramid*, which calls both functions to determine the surface area of a rectangular-based pyramid, given the two side lengths of its base and its height. Recall that the formula for a rectangular-based pyramid is

$$SA_{pyr} = (\text{area of base}) + 2(\text{area of side triangles}) + 2(\text{area of front/back triangles})$$

and that the height of a triangle making the pyramid is given by

$$\text{triangle height} = (\text{pyramid length}) \times \sqrt{\left(\frac{\text{pyramid width}}{2}\right)^2 + (\text{pyramid height})^2}$$

3. Write a function, *count_letter*, that counts the number of instances of a specific letter in a string. It should receive two arguments: the string, and the letter. For example, *count_letter*('BANANA', 'N') should return 2. Write a second function, *count_vowels*, that counts the number of individual vowels in a string. It should call your *count_letter* function several times to achieve this. Assume that all letters are uppercase.

Challenge

4. When a function calls *itself*, this process is called **recursion**. Recursion is a key concept in higher-level computer science courses. A recursive function has two components to it: a *general case*, in which the solution is expressed in terms of a smaller version of itself, and one or more *base cases*, which return value(s) once a certain point is met. For example, consider the code below, which determines the sum of the values $1+2+3+\ldots+n$.

```
def sum_1toN(n):
   if n == 0:
         return n
   else:
         return n + sum_1toN(n-1)
print(int(input("N: ")))
```

Consider the case when $n=2$. Calling *sum_1toN*(2) results in a call to *sum_1toN*(1) in the `else` block, which itself results in a call to *sum_1toN*(0). This last call returns 0 (the base case), so the call to *sum_1toN*(1) returns 1+0=1 to *sum_1toN*(2), which returns 2+1=3 to the main program.

   a. Write a function that calculates $n!$ (given *n*) using recursion. Note that $1!=1$.
   b. Write a function that calculates the *n*th Fibonacci number, $F(n)=F(n-1)+F(n-2)$. What happens when you calculate *F*(10)? How about *F*(100)? *F*(1000)?