**Data Structures and Algorithms: Recursion**

When you hear the terms recursion or recursive, this might remind you of the terms repetition and repetitive—and this is a good connection, because recursion does indeed involve repetition. However, recursion isn't just about repetition. With recursion, we solve a problem by first solving smaller instances of the same problem. In practice, this often involves calling a function from within itself—in other words, we feed some input into the function, and the function produces some output—which we then feed back into the same function. And we continue to do this until we arrive at the solution.

The idea here is to create a function that call itself. Following is the basic example of recursion.

**What is Call Stack?**

When we use functions in our code, the computer makes use of a data structure called a call stack. As the name suggests, a call stack is a type of stack—meaning that it is a Last-In, First-Out (LIFO) data structure.

So, it's a type of stack—but a stack of what, exactly?

Essentially, a call stack is a stack of frames that are used for the functions that we are calling. When we call a function, say print\_integers (5), a frame is created in memory. All the variables local to the function are created in this memory frame. And as soon as this frame is created, it's pushed onto the call stack.

The frame that lies at the top of the call stack is executed first. And as soon as the function finishes executing, this frame is discarded from the call stack.

def add (num\_one, num\_two):

output = num\_one + num\_two

return output

result = add (5, 7)

print(result)

# result 12

If we run this code in Python tutor website http://pythontutor.com/, we can get a nice visualization of what's happening "behind the scenes" in memory: