Definition

A function is defined as recursive if it calls itself somewhere inside of its code. The main aspects of recursive functions is that they repeat until you reach a base criteria, they have a maximum depth of 1000 self calls, and become dramatically slower the more calls you attempt to make. This is due to the fact that your function does not remember all of the information that it generated on its 900th call, for example, so starts all over again when making its 901st call. After we begin some exercises, you will begin to see why this is the case.

Example

An easy example to visualize would be a program that takes the factorial(!) of a number. A factorial is defined as a number times each of the integers below it until it reaches one. For example, 5! Would be equal to 5 * 4 * 3 * 2 * 1. Attempt to create a function that does this, and remember that the lowest number we will ever need is 1. After this is finished, it should look something like this:

```
def factorial(number):
    if number == 1:
        print(number, " = ", end=" ")
        return(1)
    else:
        print(number, " * ", end=" ")
        return number * factorial(number-1)


print(factorial(int(input('Which number would you like the factorial of?  '))))
```

If we were to use 5 in this example, the logic would go as follows:

5 is under the else statement, so we have 5 * (factorial(4)). The number is still not one, so we have 5 * (4 * factorial(3)). This logic ends up at 5 * (4 * (3 * (2 * (1)))). As you can see by the number of parenthesis involved, this creates a very nested statement. This is a relatively simple example, so the constant checking of whether or not the number is one will not slow down the program very much, however, this next example will make it more apparent.