Recursion

Introduction

In normal procedural languages, one can go about defining functions and procedures, and 'calling' these from the 'parent' functions. I hope you already know that. Some languages also provide the ability of a function to call itself. This is called Recursion.

Factorial

Factorial is a mathematical term. Factorial of a number, say n, is equal to the product of all integers from 1 to n. Factorial of n is denoted by n! = 1x2x3...x n.
Eg: 10! = 1x2x3x4x5x6x7x8x9x10

The simplest program to calculate factorial of a number is a loop with a product variable. Instead, it is possible to give a recursive definition for Factorial as follows:

1) If n=1, then Factorial of n = 1
2) Otherwise, Factorial of n = product of n and
Factorial of (n-1)
Check it out for yourself; it works. The following code fragment (in C) depicts Recursion at work.

Code: CPP

```cpp
int Factorial(int n)
{
    if (n==1)
        return 1;
    else
        return Factorial(n-1) * n;
}
```

The important thing to remember when creating a recursive function is to give an 'end-condition'. We don't want the function to keep calling itself forever, now, do we? Somehow, it should know when to stop. There are many ways of doing this. One of the simplest is by means of an 'if condition' statement, as above. In the above example, the recursion stops when n reaches 1. In each instance of the function, the value of n keeps decreasing. So it ultimately reaches 1 and ends. Of course, the above function will run infinitely if the initial value of n is less than 1. So the function is not perfect. The n==1 condition should be changed to n<=1.

Imagination is a very hard thing. Imagination of Recursion is all the more tricky. Think of clones. Say you have a machine to make clones of yourself, and (for lack of a better pass-time) decide to find the factorial of a number, say 10, using your clones. So, being smart, this is what you do:

First, there's only You. Let's call you You-1. You have the number 10 in your pocket. Being smart, you know that all you need to find the factorial of 10 (10*9*...*2*1) is to somehow obtain the value of 9 factorial (9*8*..*2*1), and then just multiply it with 10. So that's what you do. You turn on your machine and out pops a clone! You give the clone You-2 strict instructions to find the factorial of 9 and make it quick! Your job is done for a while, so you (You-1) stretch on your sofa sipping on your lemonade. Meanwhile...

You-2 is (you guessed it) just as smart as you! He tucks his number (9) into his pocket, turns on the machine, and out pops a clone (You-3). The new clone is given the job of 8-factorial, which it proceeds to do while (unbeknownst to you) You-2 is sipping on his own glass of lemonade on his own sofa. And so the story goes on until finally one fine day...

Out pops You-10 who is given strict instructions (by You-9) to get the factorial of 1. Now, You-10, being just as smart as any of the other you's, knows very well that the factorial of 1 is... 1. So he says to You-9 (who was just about to doze off on his sofa), "Here's your factorial of 1." You-9 snatches the result from his subordinate You-10, takes out his plasma gun, and zaps You-10 out of existence. He scribbles on a piece of paper, calculating the product of the value he got from You-10 with the number in his pocket, 2. "Heh, heh, heh" he thinks, and goes to his boss, You-8, saying,"Here's your factorial of 2..." ...blah...blah... and finally You-2 wakes you up from your slumber, and says to you, "Here's your factorial of 9" You zap him off, multiply by the 10 in your pocket, and There You Have It !! Now, wasn't that simple?

Here, 'You' were the function. The 'clones' are merely new instances of the same function. They all think and act alike. At one point, there are 10 You's (which occupies a lot of memory space). As soon as an instance returns a value and finishes its job, it is zapped off from memory.

Recursion can get much, much trickier than that - get your fundas right.

Source: http://www.go4expert.com/articles/recursion-t137/