Recursion

This lesson introduces the concept of recursion and how it is implemented in C#

WE'LL COVER THE FOLLOWING

Introduction

Recursion can be defined as:

> A *method* that calls itself until a specific *condition* is met.

Example

An excellent and simple example of *recursion* is a method that will get the **factorial** of a given number:

```
using System;

class FactorialExample
{
    static void Main()
    {
        Console.WriteLine("Factorial of 4 is: {0}",Factorial(4));
    }
    public static int Factorial(int number)
    {
        if(number == 1 || number == 0) // if number equals 1 or 0 we return 1
        {
            return 1;
        }
        else
        {
            return number*Factorial(number-1); //recursively calling the function if n is oth
        }
```

```
    }
}
```

Code Explanation

In the above example, we can see that the *method* will take an argument, `number`.

Step by step:

Given the example, executing `Factorial(4)`

1. Is `number (4) == 1`?
2. **No**? *return* `4 * Factorial(number-1) (3)`
3. Because the *method* is called once again, it now *repeats* the **first** step using `Factorial(3)` as the new argument.
4. This continues until `Factorial(1)` is executed and `number (1) == 1` returns `1`.
5. Overall, the calculation "builds up" `4 * 3 * 2 * 1` and finally returns `24`.

Many times, a *recursive* solution to a problem is very easy to program.

- The drawback of using *recursion* is that there is a lot of *overhead*.

Every time a function is called, it is placed in *memory*. Since you don't **exit** the `Factorial` function until **n** reaches **1**, **n** functions will reside in *memory*. This isn't a problem with the simple `Factorial(4)`, but other functions can lead to serious memory requirements.

The key to understanding **recursion** is that the *method* calls a new instance of itself. After *returning*, the execution of the *calling* instance continues.

This marks the end of this chapter. Enjoying learning new concepts so far? Read on to the next chapter to learn more about C#.