

# Refactoring Factorial

by Alberto Manuel Simões  
ams@di.uminho.pt



You probably know the factorial function. It is a common mathematical function defined for the integer  $n$  as the product of all the integers in the range  $[1..n]$ . Looking on common algorithms books it is easy to find one or two different algorithms to compute the factorial.

In this article I present different solutions to compute the factorial function using different aspects of the Perl language, but also talk about the performance.

## ■ Solution 1: Recursive is beautiful -----

The factorial function is habitually defined recursively. The idea is easy: if  $n = 1$ , I return 1 as a special case, otherwise, I return the product of  $n$  with the factorial of  $n-1$ :

```
sub factorial {
    my $v = shift;
    if ($v > 1) {
        return $v * factorial($v-1);
    } else {
        return 1;
    }
}
```

Some Perl linguists might prefer a solution that is easier to read in English. Instead of the C-like if-else, I use separate statements to handle the special case of 1 and all of the other cases:

```
sub factorial {
    my $v = shift;
    return 1 if $v == 1;
    return $v * factorial($v-1);
}
```

Being written in Perl I can remove the return statements, and get a more functional-like solution:

```
sub factorial {
    my $v = shift;
    if ($v > 1) {
        $v * factorial($v-1)
    } else {
        1
    }
}
```

Also, the ternary operator can my life a little easier by removing most of the syntax from the if-else:

```
sub factorial {
    my $v = shift;
    return $v > 1
        ? $v * factorial($v-1)
        : 1;
}
```

This can be a very slow operation though. To compute the factorial of  $n$ , I have to make  $n-1$  subroutine calls, and I have to do that every time I want to make the computation.

## ■ Solution 2: Iterative is faster -----

When discussing the benefits of iterative or functional programming languages, it is commonly said that the recursive solution is slower than the iterative one, so reducing the number of function calls should make it faster:

```
sub factorial {
    my $v = shift;
    my $res = 1;
    while ($v > 1) {
        $res *= $v;
        $v--;
    }
    return $res;
}
```

Other languages might be able to optimize code written in a recursive fashion to make it iterative, which is why recursion appears faster in other languages. The code that we see isn't always exactly what the computer is actually doing.

Again, Perl linguists have their own preferences. Instead of a while loop maybe they want to use a for loop:

```
sub factorial {
    my $v = shift;
    my $res = 1;
    $res *= $_ for (2 .. $v);
    return $res;
}
```