Study of an Example:

Now, let's discuss a practical problem which can be solved by using recursion and understand its basic working and see how we apply our rules. An everyday example: factorial!

To calculate factorial of n we apply this formula :

n! = n x (n-1) x (n-2) x . . . x 2 × 1

So the first rule, what is the recursion condition here ? We can ask a question, "How we calculate factorial of n-1 ?" that is right like this : n-1! = (n-1) x (n-2) x . . . x 2 × 1

And that means that can we change the first formula now : n! = n x (n-1)!

We have here our recursion as we can do the same thing with n-1!, n-2! ...

And remember

$$0!=1$$

Study of an Example:

The Second rule, what is the base case here ? We know that both factorial of 0 equals 1, this is our base case. Now, is this case can be enough as stop condition ?

Yes, because the recursion that we found is decreasing n in each call, and n must be null or positive to calculate the factorials. This means that whatever n we choose to calculate factorials of will always converge to base case.

```
FUNCTION fact(n) : INTEGER
VAR
    results : INTEGER;
BEGIN
    IF (n = 0) THEN
        results := 1; // results of the factorial is 1
    ELSE
        results := n * fact(n-1); // n! = n * (n-1)!
    END_IF
    RETURN results ;
END
```

Execution:

Calculation of 4! :

fact(4) ⇒ 4* fact(3) ⇒ 4 ∗ 3* fact(2) ⇒ 4 ∗ 3 ∗ 2 *fact(1) ⇒ 4 ∗ 3 ∗ 2 ∗ 1 * fact(0) ⇒ 4 ∗ 3 ∗2 ∗ 1 ∗ 1

= 24

Examples of FALSE programs!

No stopping conditions ⇒ INFINITE CALCULATION

```
BEGIN
        results := n * fact(n-1);
    RETURN results ;
END
```

No simplification of the case to be treated

```
BEGIN
    IF (n = 0) THEN
        results := 1;
    ELSE
        results := n * fact(n+1);
//the function is not directed to the stop condition.
    END_IF
```

```
    RETURN results ;

END
```