**Recursion That Returns a Value**

The mechanics of recursion are clearer when the recursive function returns a value instead of just performing an action, and so we examine one in detail.

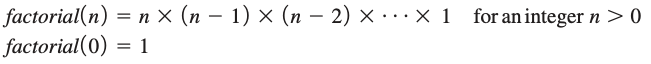
**A Recursive Valued Function: The Factorial of *n*** (commonly written as *n*!)

Note: Because the problem of factorial(n) has a simple and efficient iterative solution, you should not use the recursive solution in practice (applies to other simple algorithms as well).

**An iterative solution:**

The factorial of a negative integer is undefined.

You can iteratively loop over n, subtracting by one over each iteration, and multiply a total sum by the current value of n and its current value.



**A recursive solution:**

To define *factorial*(*n*) recursively, you first need to define *factorial*(*n*) in terms of the factorial of a smaller number.

To do so, we observe that the factorial of *n* is equal to **the factorial of n multiplied by (*n* – 1)**

Text

Description automatically generated

This definition of *factorial*(*n*) is recursive because we define the problem space of n as smaller and smaller sub-problems until we reach a base case.

In other words, this definition implies that

we can solve the problem of n! in terms of the factorial of n multiplied by (*n* – 1),

we can solve the factorial of (*n* – 1)! in terms of (*n* – 1) multiplied by (*n* – 2),

we can solve the factorial of (*n* – 2)! in terms of (*n* – 2) multiplied by (*n* – 3),

… and so on

**Adding a Base Case**

The **base case** for the factorial function is *factorial*(0), which you know is 1.

We constrain *n* to be greater than or equal to zero and each call to *factorial* decrements *n* by 1

With this definition, you will always reach the base case.

With the addition of the base case, the complete recursive definition of the factorial function is

Text, letter

Description automatically generated

**Example: factorial(4)**

*factorial*(4) = 4 \* *factorial*(3)

*factorial*(3) = 3 \* *factorial*(2)

*factorial*(2) = 2 \* *factorial*(1)

*factorial*(1) = 1 \* *factorial*(0)

*factorial*(0) = 1

At this point, the recursion stops at the base case, and we now have all of necessary information to answer the original question, so we make our way back down the call stack:

Because *factorial*(0) = 1 , *factorial*(1) = 1 \* 1 = 1

Because *factorial*(1) = 1 , *factorial*(2) = 1 \* 1 = 1

Because *factorial*(2) = 1 , *factorial*(3) = 1 \* 1 = 1

Because *factorial*(3) = 1 , *factorial*(4) = 1 \* 1 = 1