

**Introduction**

Recursion means iteration. A function is called recursive, if the body of function calls the function itself until the condition for recursion is true. Thus, a Python recursive function has a termination condition.

**Why does a recursive function in Python has termination condition?**

Well, the simple answer is to prevent the function from infinite recursion.

When a function body calls itself with any condition, this can go on forever resulting an infinite loop or recursion. This termination condition is also called base condition.

**Is there any special syntax for recursive functions?**

The answer is **NO**.

In Python, there is no syntactic difference between functions and recursive functions. There is only a logical difference.

**Advantages of Python Recursion**

1. Reduces unnecessary calling of function, thus reduces length of program.
2. Very flexible in data structure like stacks, queues, linked list and quick sort.
3. Big and complex iterative solutions are easy and simple with Python recursion.
4. Algorithms can be defined recursively making it much easier to visualize and prove.

**Disadvantages of Python Recursion**

1. Slow.
2. Logical but difficult to trace and debug.
3. Requires extra storage space. For every recursive calls separate memory is allocated for the variables.
4. Recursive functions often throw a Stack Overflow Exception when processing or operations are too large.

**Python Recursion: Example**

Let’s get an insight of Python recursion with an example to find the [factorial](https://www.mathsisfun.com/numbers/factorial.html)of 3.

3! = 3 \* 2! = 3 \* (2 \* 1!) = 3 \* 2 \* 1

This is how a factorial is calculated. Let’s implement this same logic into a program.

#recursive function to calculate factorial

def fact(n):

 """ Function to find factorial """

 if n == 1:

 return 1

 else:

 return (n \* fact(n-1))

print ("3! = ",fact(3))

**Output**

3! = 6

This program can calculate factorial of any number supplied as the argument.

### ****Explanation of the program****

First is a base condition in any recursive function. If the value of n is equal to 1, then the function will return 1 and exit. Till this base condition is met, the function will be iterated.

In the program above, the value of the argument supplied is 3. Hence, the program operates in following way.



When the function is called with the value of n

**In recursion 1**: Function returns 3 \* fact(2). This invokes the function again with the value of n = 2.

**In recursion 2**: Function checks if n = 1. Since it’s False function return 3 \* 2 \* fact(1). This again invokes the function with the value of n = 1.

**In recursion 3**: Function checks if n = 1. This returns True making the function to exit and return 1.

Hence after 3 recursions, the final value returned is 3 \* 2 \* 1 = 6.

### ****Is there any limit on the number of recursions for a Python recursive function?****

The answer is **YES**.

Unless we explicitly set the maximum limit of recursions, the program by default will throw a Recursion error after 1000 recursions.

what happens when we supply 1100 as an argument in Python recursive function and the program have to call itself recursively over 1000 times.



To avoid such errors, we can explicitly set recursion limit using sys.setrecursionlimit( ).

import sys

sys.setrecursionlimit(1500)