Recursion

process in which a function calls itself

Mohd Amir khan

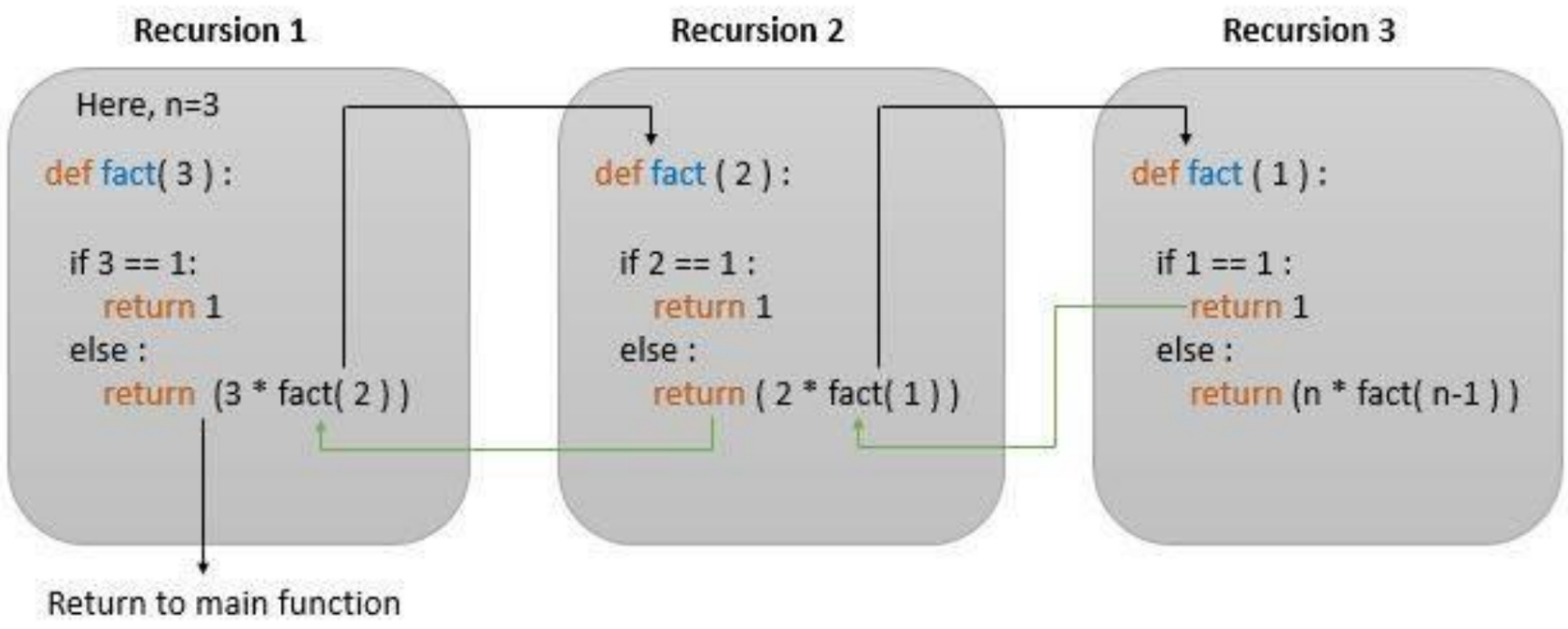
Technical Trainer

GLA University, Mathura

What is Recursion ?

- Recursion is a computer programming technique involving the use of a procedure, subroutine, function, or algorithm that calls itself in a step having a termination condition
- Solving a Python problem iteratively might include using a for or while loop. These are some of the most common tools used for incremental problem solving in any coding language. Loops are great because, a lot of the time, their implementation is intuitive and straightforward.
- In other words, a recursive function is a function that calls itself until a “base condition” is true, and execution stops. The recursive function has two parts:

1. Base Case
2. Recursive Structure

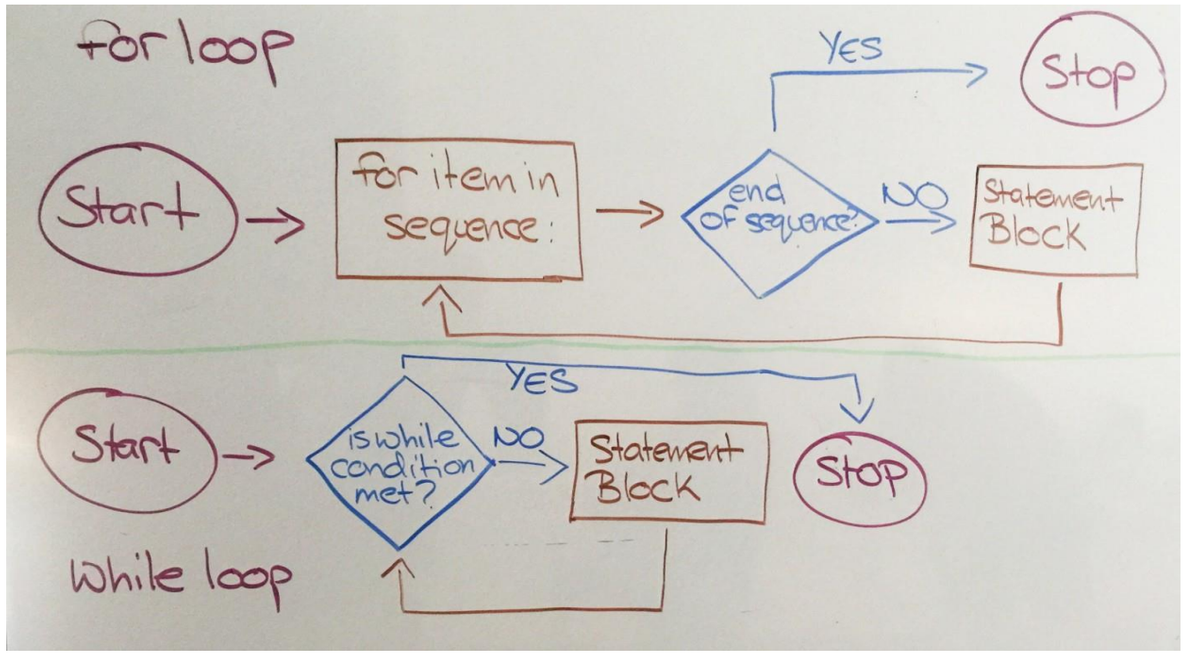


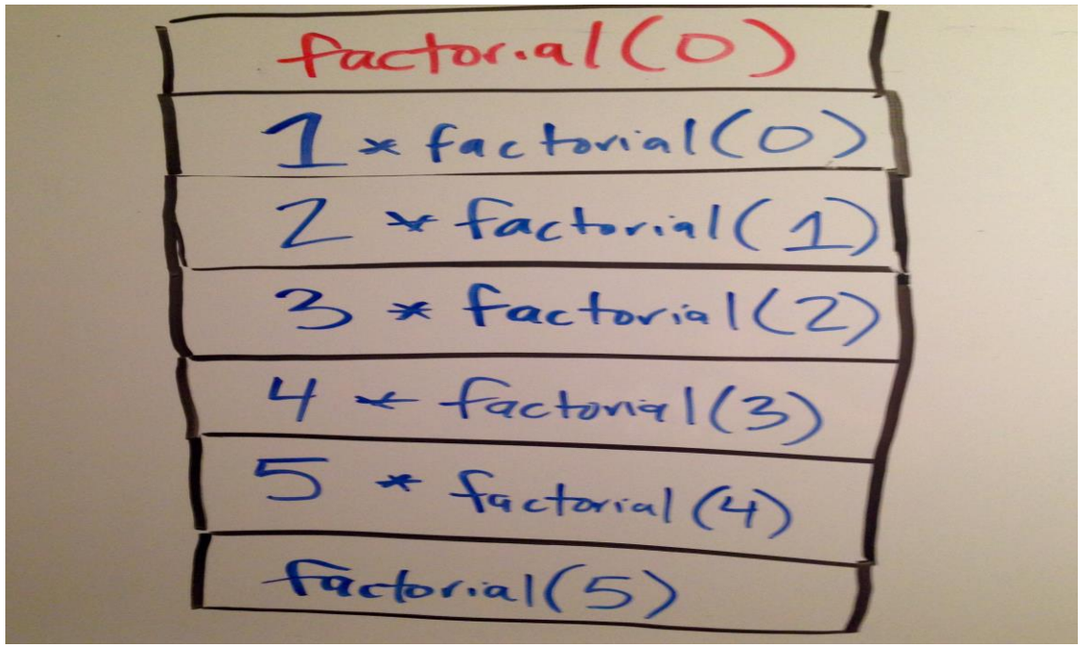
When to loop? When to recurse?

A big difference between recursion and iteration is the way that they end. While a loop executes the block of code, checking each time to see if it is at the end of the sequence, there is no such sequential end for recursive code.

For loops, like the one in the example above, iterate over a sequence. We generally use for loops when we know how many times we need to execute a block of code. In the above example, we only need to do lines 2 and 3 for the number of names in name_list.
We might also need to loop for an undetermined number of times or until a certain condition is met. This might be a good time to use a while loop.

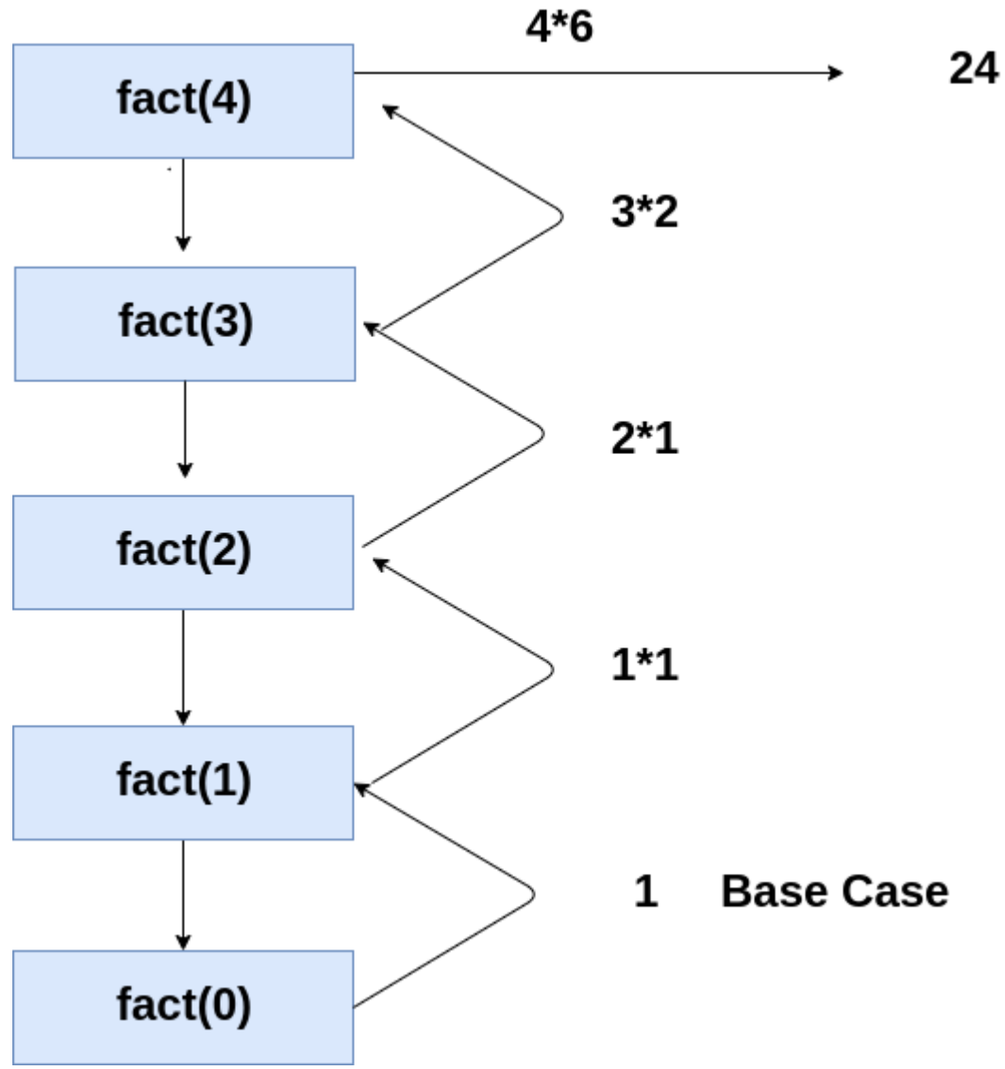
Recursion is made for solving problems that can be broken down into smaller, repetitive problems. It is especially good for working on things that have many possible branches and are too complex for an iterative approach.
One good example of this would be searching through a file system. You could start at the root folder, and then you could search through all the files and folders within that one. After that, you would enter each folder and search through each folder inside of that





Factorial Example

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n*fact(n-1)
```



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.

Is infinite recursion is infinite?
- Answer is No
- Infinite recursion in programming is never "infinite" in practice.

this is because each recursive function call takes up new stack space, and the system is bound to run out of memory and crash soon

Terminology :

Recursive function

A function that calls itself!

Base Case:

The base case returns a value without making any subsequent recursive calls
for factorial(), the base case is n = 1

Recursive structure

A data structure that is partially composed of smaller or simpler instances of the same data structure

Recursion Examples…

01 Count Down:-

```
countdown(5)
print("happy    recursion   day")
```

```
5
4
3
2
1
happy recursion day
```

Countdown Algorithm

- If I want to countdown from 0,
  then it's over! Don't do anything.
- If I want to countdown from N (>0),
  then count N, and countdown from N-1.

Countdown Example

```
def countdown(n):
    if n == 0:
        return
    print(n)
    countdown(n - 1)

countdown(5)
print("happy recursion day")
```

02 :GCD of two Number

```
def  gcd(a,  b):
      if  a < b:
            return  gcd(a,  b - a)
      elif  b < a:
            return  gcd(a - b,  b)
      else:
            return  a

print(gcd(68, 119))
```

Output

-- 68 119

-- GCD(68, 119) is 17

Why Recursion?

- Sometimes it's easier to write and read code in recursive form.
- You can always convert an iterative algorithm to recursive and *vice versa*.
  (does not mean it's easy)

Reverse String Example

```
def reverse(str):
    if str == "":
        return str      # or return ''
    else:
        return reverse(str[1:]) + str[0]

print(reverse("reenigne"))
```

Sum of Digits

```
print(sum_digits(314159265))  # => 35
```

- **NO LOOPS!**
- **NO STRINGS!**

Recursive Algorithm

- **Sum of digits of 0 is 0.**
- **Sum of digits of N >0:**

  Find last digit + sum of digits except last.

  N % 10

  N // 10

Sum of Digits

```
def  sum_digits(n):
      if   n  == 0:
            return   0
      else:
            return   (n  %10)  +  sum_digits(n   //   10)

print(sum_digits(314159265))
```

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

Practice Examples

Q1. Write a Python program of recursion list sum. Test Data: [1, 2, [3,4], [5,6]]
Expected Result: 21

Q2. We can define the sum from 1 to x (i.e. 1 + 2 + ... + x)
recursively as follows for integer x ≥ 1:
1, if x = 1
x + sum from 1 to x-1 if x > 1

Complete the following Python program to compute the sum 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 recursively:

```
def sum(x):
        # you complete this function recursively

num = int(input())
print(sum(10))
```

MCQ (Click Here)