Why Is Recursive Function So Difficult to Understand

I came across this situation a couple times that a function needed to be recursive. Once I had to simulate the recursion in VBA because at that time calling itself was not allowed. I basically constructed a simulated call stack.

Before I go further, I would like to share what I have experienced with programming. Almost all programming is about handling of data and calculation. The harder part for this kind of process is examining the state of data with nested if-then-else logic structure. I once had a logic structure four layer deep because there are many different intertwined conditions when all different pieces of data come together and decision must be made on how to process them based on the combination of the conditions.

A Word About Computer Programming Languages

As I just mentioned, a lot of the programming is about handling data. A data storage is opened or connected to. New data is added to it. Data in it may be updated or deleted. Data may be moved to other storage. Data may be used to produce reports. Data may be used to determine the price of new car model, etc.

Any programming language can easily handle the tasks mentioned above. However, I am not talking about programming yet. Rather I want to point out that one of magical thing programming language does is that it allows data to be organized in infinitely complex way. Without this, we would still be using printed maps. Artificial intelligence would have never been invented. No video games, etc. Of course, mathematics was the source of inspiration. It helps us explore new possibilities. Computer language is the tool to turn possibilities into reality.

Recursion is the other great feature of programming language that helps us solve some problems in a simple and elegant way.

What Makes Recursions Different and Difficult

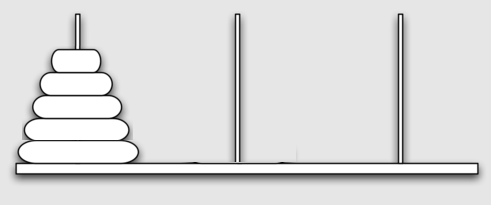
Now I am going to talk about recursions and programming.

Let me started by writing down few base points about regular functions:

* Normally, a regular function is created to perform a task. The process and the resources can be visualized. Opening a file is very similar to opening a book. Adding data is like writing.
* inside every function, **other** functions may be called to help it perform part of the tasks, like establishing connection to database.
* The kind of mathematics used in the functions when it runs is usually related to arithmetic and calculus.
* It creates local data structure.

Recursive functions have none of above.

* Recursion is very difficult be visualized. At least not in the way mentioned above. It is somewhat abstract.
* It calls itself while it is still being put together which is counter-intuitive.
* It involves very little arithmetic and calculus. In fact, it handles the complexity of the problem by way of recursion itself. Of course, understanding how it works is where the obstacle lies.
* If there is a non-recursive way, including recursion simulation, to solve the problem, local data structure needs to be created. A recursive function should have no local variables in it. All data involved are parameters for the function. At least that’s my experience so far.



The Tower of Hanoi

This function is a great example on how recursion presents such a small beautiful solution to a seemingly complex problem. It solves the problem by doing recursion 95% of the time.

In this problem there are three poles, A, B and C. Say 5 round discs with different sizes and holes in the middles of them are stacked on pole A in a way that each disc is smaller than all others below it. The goal is to move them to pole C.

There are rules governing how discs should be moved:

1. Only one disc can be moved at a time.
2. Only the top-most disc can be moved at any time.
3. On any pole, no disc is bigger than others below it at any time.

In my next article, I will illustrate a way to understand how this small recursive function work to solve this problem. Please review the towerOfHanoi function and read the last section again.

Here is the script:

tower = {

'A': [5,4,3,2,1],

'B': [],

'C': [],

}

# print the tower in horizon direction

# put this function in different spot in towerOfHanoi to visualize the placement of discs

def printTower(): # print the tower in horizon direction

global tower

for t in 'ABC':

print(f"<{t}> ", end='')

for d in tower[t]:

print(f"{d} ", end='')

print('')

# the recursive function

def towerOfHanoi(n, s\_pole, i\_pole, d\_pole):

global tower

if n == 1:

tower[d\_pole].append(tower[s\_pole][-1])

del tower[s\_pole][-1]

# print(f"<1> {s\_pole} ---> {d\_pole}”)

else:

towerOfHanoi (n-1, s\_pole, d\_pole, i\_pole)

tower[d\_pole].append(tower[s\_pole][-1])

del tower[s\_pole][-1]

# print(f"<{n}> {s\_pole} ---> {d\_pole}”)

towerOfHanoi (n-1, i\_pole, s\_pole, d\_pole)