Recursion example: Tower of Hanoi

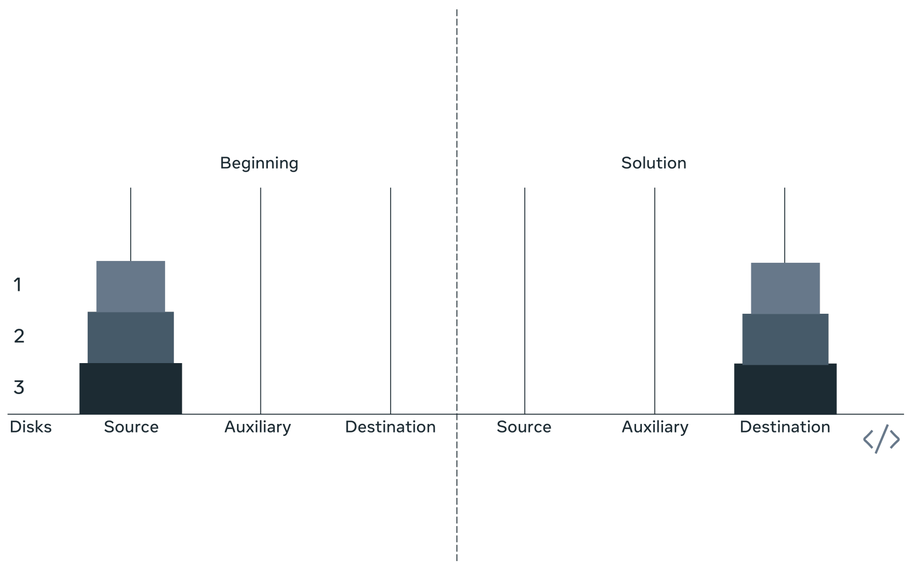
Introduction

The Tower of Hanoi is a popular puzzle in mathematics and programming. It's widely considered as a good way to demonstrate recursion. There is a legend about an ancient Hindu temple containing a large room with three posts surrounded by 64 golden disks. The puzzle was presented to a young priest who was told to move all the disks from one post to another without violating a set of given rules. By estimates, if the priest followed the rules and moved one disk per second, the puzzle would be solved in 2^64 – 1 second. That's about 585 billion years. By then the temple would no longer exist. Inspired by this legend, a French mathematician Edouard Lucas invented the Tower of Hanoi puzzle more than a hundred years ago.

Objective and rules of the puzzle

The objective is to move **n** number of disks from one tower to another following a set of rules. These rules are as follows:

- Only one disk can be moved at a time
- Only the upper disk of any of the towers can be moved
- Larger disks cannot be placed over smaller disks



An image of the initial and final stages of the puzzle is provided above.

In the optimal scenario of solving the puzzle, the total moves will be 2^n – 1 where n is the number of disks that need to be moved.

Now let's examine how you can write code for this in Python using the principles of recursion that you have learned.

Understanding codeblocks

You begin with three towers or poles, source destination and helper. In the first section of code, you will cover the base condition of recursion. Base conditions serve primarily to complete the execution and ensure the recursion does not run into an infinite loop.

The second section of code deals with the actual call to the recursion function within itself.

The third section consists of the driver code for the initial call, consisting of the actual tower names that you pass as arguments to the function, along with the number of disks. Driver code is a generic term used to denote the section of code that gives the actual call to the function or class.

```
# Recursive function for Tower of Hanoi
def hanoi(disks, source, helper, destination):
    # Base Condition
    if (disks == 1):
        print('Disk {} moves from tower {} to tower {}.'.format(disks, source, destination))
        return

    # Recursive calls in which function calls itself
    hanoi(disks - 1, source, destination, helper)
    print('Disk {} moves from tower {} to tower {}.'.format(disks, source, destination))
    hanoi(disks - 1, helper, source, destination)

# Driver code
disks = int(input('Number of disks to be displaced: '))
'''
Tower names passed as arguments:
Source: A
Helper: B
Destination: C
'''
# Actual function call
hanoi(disks, 'A', 'B', 'C')
```

Output

```
Number of disks to be displaced: 3
Disk 1 moves from tower A to tower C.
Disk 2 moves from tower A to tower B.
Disk 1 moves from tower C to tower B.
Disk 3 moves from tower A to tower C.
Disk 1 moves from tower B to tower A.
Disk 2 moves from tower B to tower C.
Disk 1 moves from tower A to tower C.
```

Explanation

If you can imagine the disks in question as displayed in the image, you can understand how the code successfully displaces the three disks from tower A to C, following the expected rules.

Note how the variable disk initially takes the number of disks as the input and later is read or understood as the disk number in question inside the code.

In the block of code, the first section is the base condition that that you apply when using disk 1. Once executed, it returns to the rest of the execution flow out of the if condition.

The remaining disks are moved by passing the values from source to helper with destination as helper. The remaining disk is moved from source to destination. The remaining n-1 disks on the helper are moved from helper to destination with source as the helper.

In the last section, the driver code takes the input for the number of disks I want to move. In accordance, I pass the values of A, B and C as the tower names and give the function call.

You will notice that it took 2^3 - 1 = 7 steps to complete the transfer, which meets my expectations.

The Tower of Hanoi and recursion, in general, can be confusing, even for an avid Python programmer. As such, the best way to understand recursion is by inserting the values and running a dry code using a pen and paper to see how the values change and which function gets called in the code at what point.

Mark as completed

Like **Dislike** **Report an issue**