DAILY DSA | DAY-22 | Recursion – Part 2|
-GOPALKRISHNA A

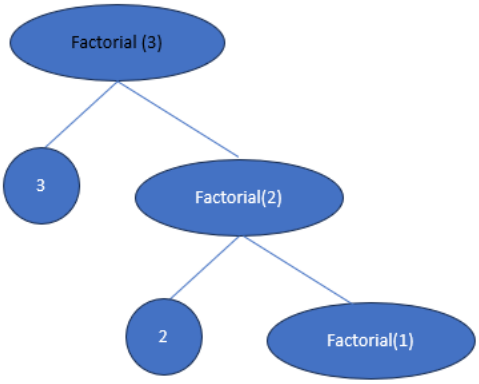
Types of recursion

Head recursion: In head recursion, the recursive call comes at the beginning of the function, and we post-process the results of the recursive call before returning. For example, consider the following function for calculating the factorial of input integer n:

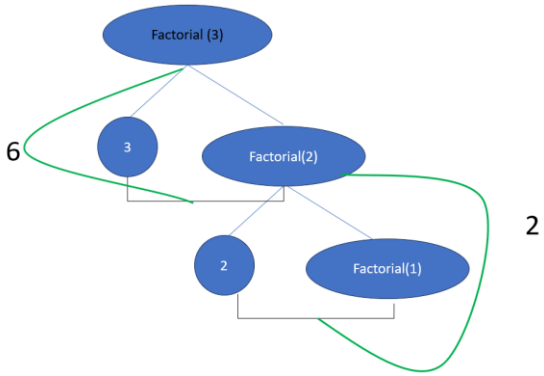
```
def factorial_head(n: int):
  output = None
  if n == 0:
    output = n
  elif n > 0:
    output = n * factorial_head(n-1)
  return output
```

Rather than directly returning the result of factorial_head(n-1), we first post-process its output by multiplying it by n before returning.

For better understanding, we visualize these recursive functions via a tree diagram that shows the flow of logic:



The above diagram shows what happens when we add recursive calls to the function call stack. Now let's draw in what happens when we start returning from the recursive calls and coming back up the function call stack:

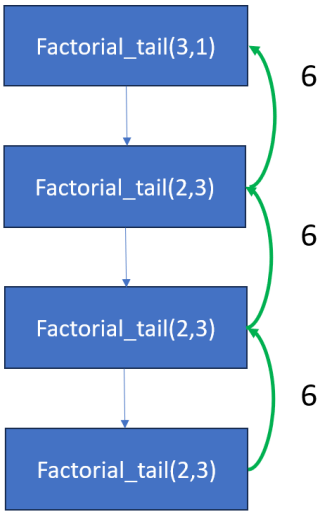


The green arrows are labeled with the output from each level of the tree.
factorial(2) = 2 x factorial(1) = 2, factorial(3) = 3 x factorial(2) = 6, and so on.

Tail recursion:
In the tail recursion, the function does its work before the recursive call and directly returns the result of the recursive call. Thus, the recursive call occurs at the tail end of the function.
Here is an example of how we can rewrite the factorial as a tail recursive function:

```
def factorial_tail(n: int, accum: int):
  if n == 0:
    return accum
  accum = accum * n
  return factorial_tail(n-1, accum)
```

As before, we can also draw the recursion tree:



Unlike in head recursion, a tail recursive function does all its processing as it's going down the recursive function call stack, and directly returns the answer of the recursive calls. This is typically more efficient than head recursion since we no longer need to store immediate variables while we wait for the results of the recursive calls.
Advantages of recursion:

- Recursion can simplify complex problems by breaking them down into smaller, more manageable sub-problems. This can make it **easier for programmers to understand** and solve problems.
- For a recursive function, we only need to define the base code and recursive based which makes it **shorter code** than an iterative code

Disadvantages of recursion:

- A recursive program has **greater space requirements** than an iterative program as each function call will remain in the stack until the base case is reached
- It also has **greater time requirements** because each time the function is called, the stack grows and the final answer is returned when the stack is popped completely.
- Stack overflow may happen if the recursive calls are not properly checked.