

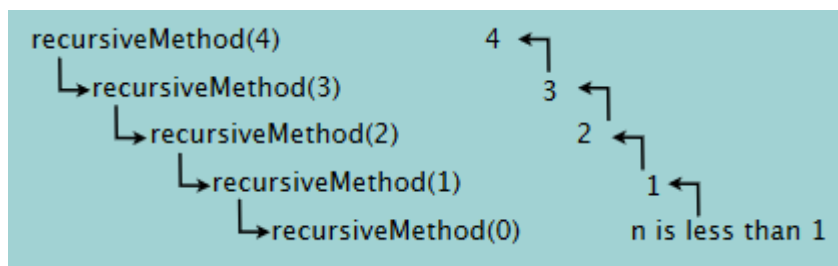
# Data Structure

# Now Will See the How Recursive method is Stored in Stack Memory.

## How Recursion works?

```
def recursiveMethod(n):  
    if n<1:  
        print("n is less than 1")  
    else:  
        recursiveMethod(n-1)  
        print(n)
```

1st



2nd

Note: See the above image where it states about how Recursion Works.

- one function is there ie; recursiveMethod() user is given input 4 so the flow of the recursive function is the same as in the above image.

```
def Loading... (n):
    if n < 1:
        print("n is less than 1")
    else:
        recursive(n-1)
        print(n)
recursive(4)
```

```
n is less than 1
1
2
3
4
```

3rd

- see the 2nd image based on the LIFO (Stack) as we can see the last method recursiveMethod(0) called. so lastmethod will pop out first. ie; n is less than 1 and so on.

Note: we understood that stack memory is used by the system for managing the recursive calls.

- So every time Recursive method calls itself, the system stores it in the stack for coming back because there are execution (print) statement left after calling itself.
- This means that system somehow remembers the point where it should stops, and call to function with different parameter. based on the condition.

## Recursive vs Iterative Solutions

```
def powerOfTwo(n):
    if n == 0:
        return 1
    else:
        power = powerOfTwo(n-1)
        return power * 2
```

```
def powerOfTwoIt(n):
    i = 0
    power = 1
    while i < n:
        power = power * 2
        i = i + 1
    return power
```

- Here we can see the two functions are given one based on the Recursion, and another based on the iterative traditional method of looping concept.

- in the Recursion function, as we can see the above image we have a one condition to stop further execution (Infinite Loop).
- if the condition is not satisfied then it will execute the else block and return the power of 2.
- Conditional statement decides the termination of Recursion.
- Here in Recursive function, infinite Recursion can leads the system crash.
- Recursion repeatedly invokes (triggering) the mechanism consequently (accordingly) as per method calls.
- so conclusion it can be Expensive for both processor time and memory Space. as we can discuss in previous section where it recursive function call stored the function in stack memory.
- That means if the algorithm resources depth of N (N is number of times so it directly depends on the number so till it will execute based on the Number. so it uses at least  $O(N)$  memory.

# in other side in Iterative solution

- we have created a two variable which is i and power, until i will less than 1 this while loop will execute till that time.
- As per analyzing the two function, Recursion code is easy to write. compare to Iterative one.
- but here variable value of i will decide the termination of execution.
- here in iterative function, infinite iteration consume CPU cycles ie; (CPU usage).
- where in iterative function while executive it will not store any instance while executing

Note: So for this reason, its better to implement the recursive Algorithm iteratively. ( which based on the loop system.)

# so the question is does iteration look always better than Recursion.?

- so the answer is NO because each iteration logic having their own Advantage,

- We use the Recursion especially in the case of when the bigger problem is divided in similar problems.
- And when we deal with tree and graph the uses of recursion is more on that concept.

## Differences between Recursion Vs Iteration.

Points	Recursion	Iteration	
Space efficient?	No	Yes	No stack memory require in case of iteration
Time efficient?	No	Yes	In case of recursion system needs more time for pop and push elements to stack memory which makes recursion less time efficient
Easy to code?	Yes	No	We use recursion especially in the cases we know that a problem can be divided into similar sub problems.

## When To Use Or Avoid a Recursion.

- Iteration performs better than Recursion in terms of time and space complexity. (its consuming the memory space )

**When to use it?**

- When we can easily breakdown a problem into similar subproblem
- When we are fine with extra overhead (both time and space) that comes with it
- When we need a quick working solution instead of efficient one
- When traverse a tree
- When we use memoization in recursion

- preorder tree traversal : 15, 9, 3, 1, 4, 12, 23, 17, 28

**When avoid it?**

- The second point describes about as we already saw recursion take a space (Memory Space) and time(Time Complexity). So if your System is ready/able to handle the both complexity then you can use Recursion.

Note: for Example suppose if your developing a mobile Application Which should run on low-memory devices as well. So in this Cases, Recursion is not Advisable.