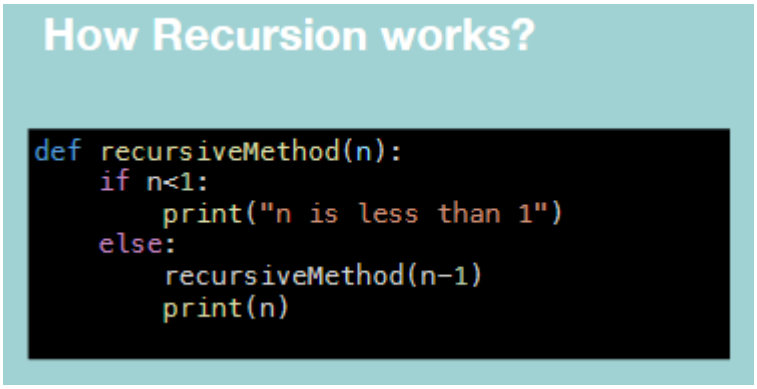
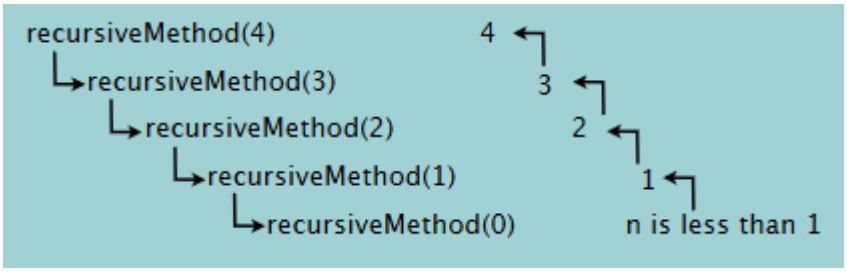
Data Structure

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



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,

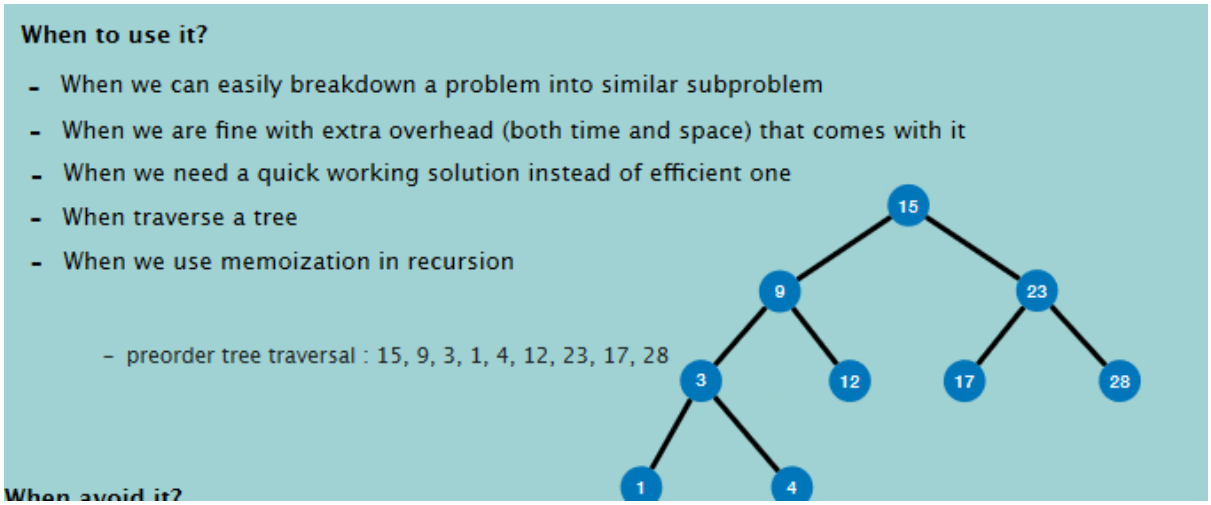
- We use the Recursion especially in the case of when the bigger problem is divided in similar problems.
- And when we deal with tree and graph the uses of recursion is more on that concept.

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)



- 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.

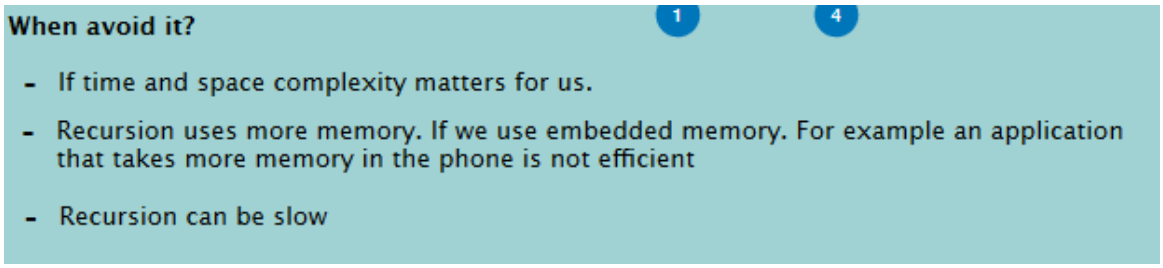
Note: for Example suppose if your developing a mobile Application Which should run on low-memory devices as well. So in this Cases, Recursion is not Advisable.

Note : suppose if you are developing an algorithm for a critical system. which should must be so fast, like Air Bag system in the car or gun bullets shots, where a fraction of speed matters a lot. so in this case avoid Recursion.

Note: Refer the third point, as we can see the writing code using Recursion is much easier. ie; Solving the mathematical problems like Fibonacci or factorial so, in this case we can use Recursion as an easy solution.

Note: Tree is a collection of an object, that are linked to one another. Recursion is very beneficial when we use pre-order traversal where we used Recursion. from the above image take a reference traversal flow.

When To Avoid a Recursion.



- The first point tells about we already saw when we use Recursion, its took time and space (for every execution the instance were stored into Stack) so its takes time and space while executing.

- Recursion using more memory because the Recursive function has to add the into the stack with each Recursive call and keep storing the values until function is not finished.
- Recursion can slow means for every call value has to stored into the stack so its time consuming process and also its based on stack (LIFO).

Recursion Writes In Optimise Way.

Write a Factorial code.

Example 1

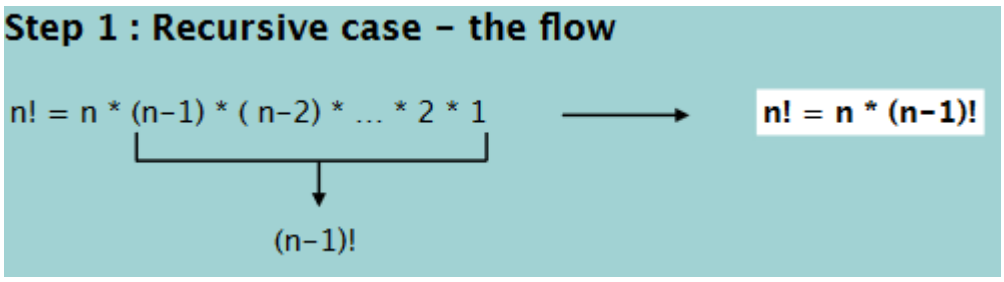
4! = 4*3*2*1=24

Example 2

10! = 10*9*8*7*6*5*4*3*2*1=36,28,800

n! = n*(n-1)*(n-2)*...*2*1

Note: Factorial of 0! is 1.



- Calculation be like the above image.

(n-1)! = (n-1) * (n-1-1) * (n-1-2) * ... * 2 * 1 = (n-1) * (n-2) * (n-3) * ... * 2 * 1

- So above the image express the expression of (n-1)! Factorial.