# Recursion (Tutorial + Notes)

Recursion is a process where a function calls itself. If you've seen the movie \*Inception\*, this concept may feel familiar. Imagine Leonardo's character having a dream, within which he has another dream, and so on. Translating this into programming, it's like having a function named `dream()` that calls itself repeatedly:

function dream()  
 print "Dreaming"  
 dream()

## Why Use Recursion?

Recursion is particularly useful for solving problems that can be divided into smaller, similar problems. Let's explore its key components and how to use it effectively.

## Key Concepts

### 1. Base Case

A base case is a condition where the function stops calling itself. Without it, the recursion would continue infinitely, leading to a stack overflow error.

For example, in calculating the factorial of a number, we know:  
  
- factorial(0) = 1  
  
So, the base case is when the input number x is 0. Here's the pseudo code:

function factorial(x)  
 if x is 0 // base case  
 return 1  
 return x \* factorial(x - 1) // recursive call

### 2. Number of Recursive Calls

There is a limit to how many recursive calls can be made due to the finite size of the stack. To avoid stack overflow:  
  
- Ensure the base case is reachable.  
- Avoid unnecessary recursive calls.

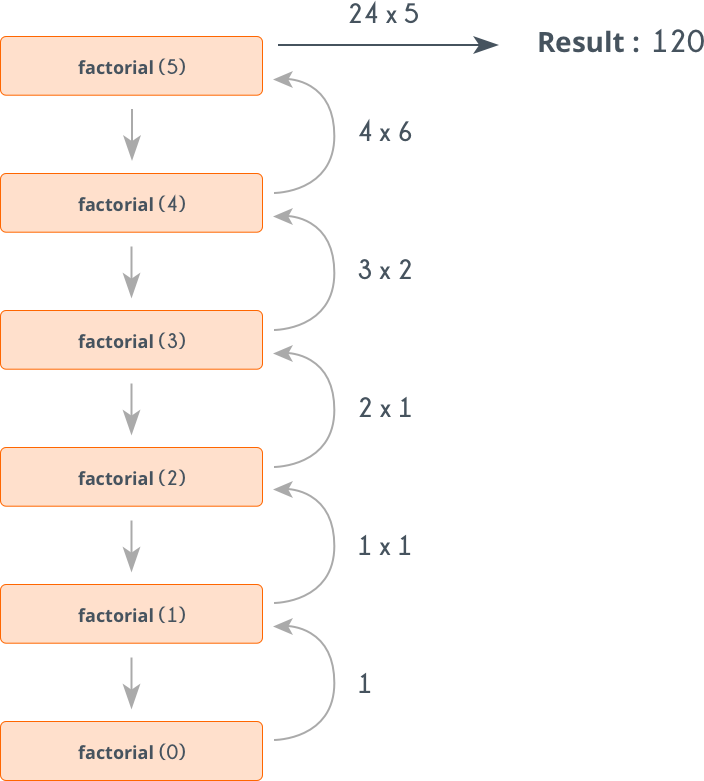
### 3. Breaking Down Problems

To solve a problem using recursion:  
  
1. Break it into smaller problems of the same type.  
2. Ensure there is a base case.  
3. Confirm that the base case is reached before exceeding the stack limit.

## Example: Factorial Using Recursion

The factorial of a number is the product of all positive integers up to that number. For example:  
  
- factorial(5) = 5 × 4 × 3 × 2 × 1 = 120

#include <iostream>  
 using namespace std;  
  
 int fact(int n) {  
 // BASE CASE  
 if (n == 0 || n == 1)  
 return 1;  
   
 return n \* fact(n - 1); // RECURSIVE CALL  
 }  
  
 int main() {  
 cout << "Factorial of 5: " << fact(5);  
 return 0;  
 }



## Visualizing Recursion

Let’s visualize how recursion works for factorial(5):

1. Call: factorial(5)  
 - Returns 5 \* factorial(4)  
 2. Call: factorial(4)  
 - Returns 4 \* factorial(3)  
 3. Call: factorial(3)  
 - Returns 3 \* factorial(2)  
 4. Call: factorial(2)  
 - Returns 2 \* factorial(1)  
 5. Call: factorial(1) (Base Case)  
 - Returns 1  
  
 Now, the function resolves step by step:  
 - factorial(1) = 1  
 - factorial(2) = 2 \* 1 = 2  
 - factorial(3) = 3 \* 2 = 6  
 - factorial(4) = 4 \* 6 = 24  
 - factorial(5) = 5 \* 24 = 120

## Important Notes

- Always include a base case to prevent infinite recursion.  
- Be mindful of the stack size limit.  
- Use recursion when the problem naturally fits the divide-and-conquer approach, like:  
 - Calculating factorials  
 - Solving Tower of Hanoi  
 - Traversing trees  
 - Searching in graphs (DFS)