**What is Recursion?**

* Any function or method which calls itself is called recursion. A recursive method solves a problem by calling a copy of itself to work on a smaller problem. This is called the recursive step.
* The recursion step can result in many more such recursive calls.
* It is important to ensure that the recursion terminates at some point.
* Each time the function calls itself with a slightly simpler version of the original problem.
* The sequence of a smaller problems must eventually converge on the base case.

**Why Recursion?**

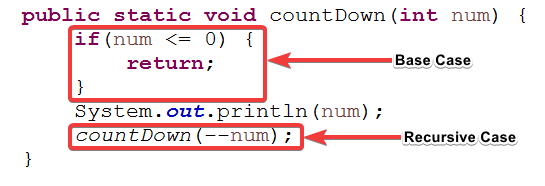
* Recursion is a useful technique borrowed from mathematics. Recursion code is generally shorter and easier to write that iterative code.
* Generally, loops are turned into recursive function when they are compiled and interpreted.
* Recursion is most useful for tasks that can be defined in terms of similar subtasks.
* For example: sort, search, and traversal problems often have simple recursive solutions.

**Before, going deeper into recursion. Let’s first discuss about functions.**

* In almost all the programming languages, there is a built in data structure that manages what happens when functions or methods are invoked. Generally, it’s called as call stack or method stack.
* It’s a **Stack** data structure. Any time a function is invoked it is placed (**pushed**) on the top of the call stack.
* When a method/function sees the return keyword or when the function ends, the compiler will remove (**pop**)

**Two essential parts of a recursive function!**

* **Base Case**
* **Recursive Case**



**Format of a Recursive Function**

* A recursive function performs a task in part by calling itself to perform the subtasks.
* At some point, the function encounters a subtask that it can perform without calling itself. This case where the function **does not recur is called as Base Case.**
* **The former, where the function calls itself to perform a subtask, is referred to as the Recursive Case.**

*if(test for the base case)*

*return some base case value;*

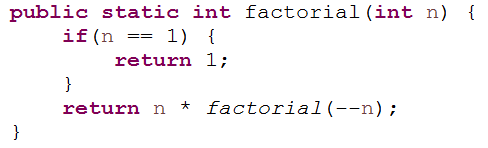
*else if(test for another base case)*

*return some other base case value;*

*//the recursive case*

*else return(some work and then recursive call);*

**Calculate factorial of a positive integer?**

****

**Recursion vs Iteration**

A recursive approach makes it simpler to solve a problem, which may not have the most obvious of answers. But, recursion adds overhead for each recursive call (needs **space on the stack frame**).

|  |  |
| --- | --- |
| **Recursion** | **Iteration** |
| Terminates when a base case is reached | Terminates when a condition is proven to be false |
| Each recursive call required extra space on the stack frame (memory) | Each iteration does not require any extra space |
| If we get infinite recursion, the program may run out of memory and give stack overflow | An infinite loop could run forever since there is no extra memory being created. |
| Solutions to some problems are easier to formulate recursively. | Iterative solutions to a problem may not always be as obvious as a recursive solution. |

**Notes on Recursion:**

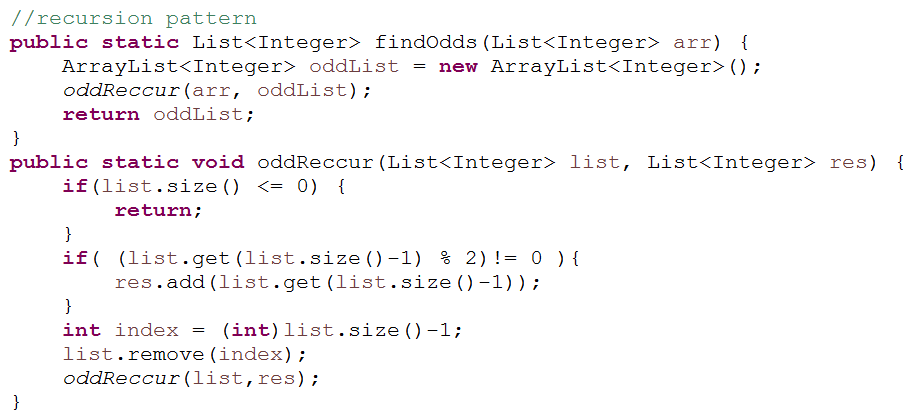
* Recursive algorithms have two types of cases: **base case** and **recursive case.**
* Every recursive function case must terminate at base case.
* Generally iterative solutions are more efficient than recursive solutions. Due to the overhead of function calls in recursion.
* A recursive algorithm can be implemented without recursive function calls using a stack. But, it’s generally more trouble than its worth. That means any problem that can be solved recursively can also be solved iteratively.
* For some problems, there are no obvious iterative algorithms.
* Some problems are best suited for recursive solutions while others are not.

**Example Algorithms of Recursion:**

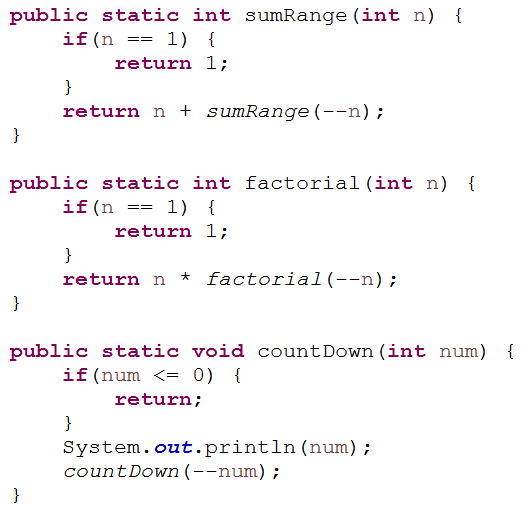
* Fibonacci Series, Factorial Finding
* Merge Sort, Quick Sort
* Binary Search
* Tree Traversal and many Tree Problems: InOrder, PreOrder, PostOrder
* Graph Traversal: DFS [Depth First Search] and BFS [Breadth First Search]
* Dynamic Programming Examples
* Divide and Conquer Algorithms
* Towers of Hanoi
* Backtracking algorithms

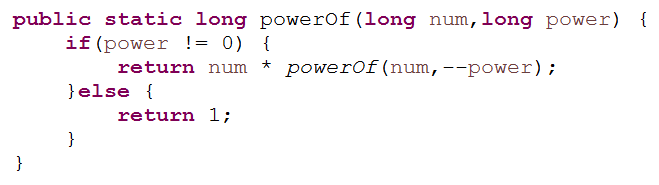
**Examples of Recursion:**

**Example 1:**

****

**Example 2:**

****

****