# Recursion Basics

**What is Recursion?**

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are

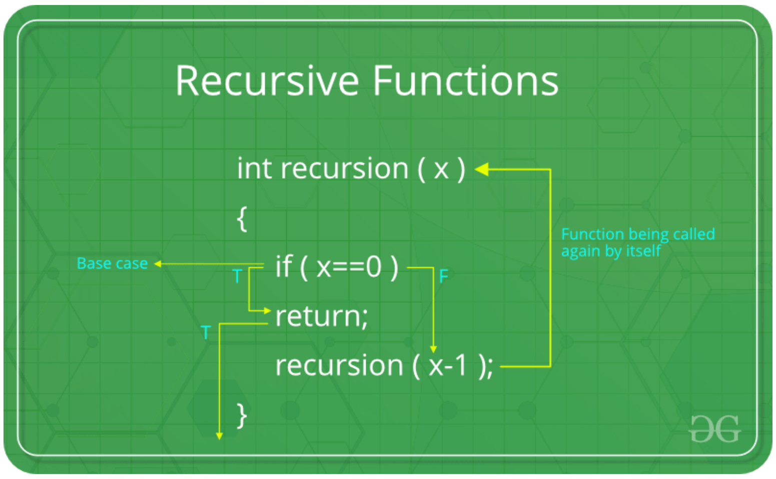
Towers of Hanoi (TOH) , Inorder/Preorder/Postorder Tree Traversals , DFS of Graph, etc.

**What is the base condition in recursion?**

In a recursive program, the solution to the base case is provided and the solution of bigger problem is expressed in terms of smaller problems.

```
int fact(int n)
{
    if (n < = 1) // base case
        return 1;
    else
        return n*fact(n-1);
}
```

In the above example, the base case for n < = 1 is defined and a larger value of a number can be solved by converting to a smaller one till the base case is reached.



```
int recursion ( x )
{
    if ( x==0 )
    return;
    recursion ( x-1 );
}
```

**How a particular problem is solved using recursion?**

The idea is to represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop recursion. For example, we compute factorial n if we know factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0.

**Why Stack Overflow error occurs in recursion?**

If the base case is not reached or not defined, then the stack overflow problem may arise. Let us take an example to understand this.

```
int fact(int n)
{
    // wrong base case (it may cause
    // stack overflow).
    if (n == 100)
        return 1;

    else
        return n*fact(n-1);
}
```

If fact(10) is called, it will call fact(9), fact(8), fact(7), and so on but the number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on the stack, it will cause a stack overflow error.

**How memory is allocated to different function calls in recursion?**

When any function is called from main(), the memory is allocated to it on stack. A recursive function calls itself, the memory for the called function is allocated on top of memory allocated to the calling function and a different copy of local variables is created for each function call. When the base case is reached, the function returns its value to the function by whom it is called and memory is de-allocated and the process continues. Let us take the example of how recursion works by taking a simple function:

```
void printFun(int test)
{
    if (test < 1)
        return;
    else
    {
        print test;
        printFun(test-1);    // statement 2
        print test;
        return;
    }
}
// Calling function printFun()
int test = 3;
printFun(test);
```

**Output**:

```
3 2 1 1 2 3
```

When

**printFun(3)**

is called from main(), memory is allocated to

**printFun(3)**

and a local variable test is initialized to 3 and statement 1 to 4 are pushed on the stack as shown in below diagram. It first prints ‘3’. In statement 2,

**printFun(2)**

is called and memory is allocated to

**printFun(2)**

and a local variable test is initialized to 2 and statements 1 to 4 are pushed in the stack. Similarly,

**printFun(2)**

calls

**printFun(1)**

and

**printFun(1)**

calls

**printFun(0)**

.

**printFun(0)**

goes to if statement and it returns to

**printFun(1)**

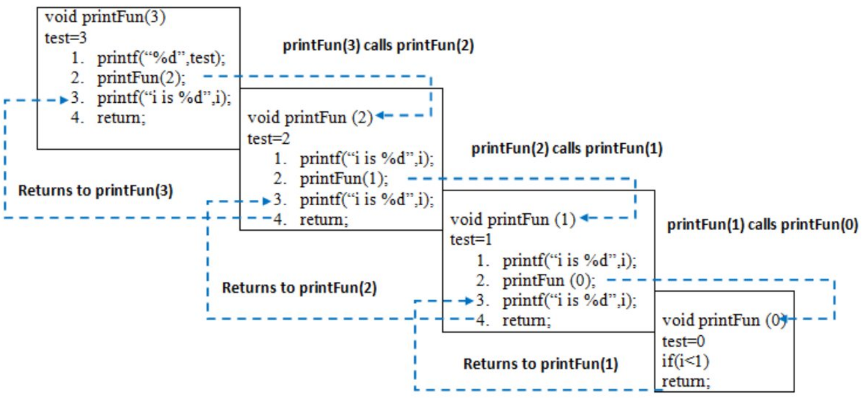
. Remaining statements of

**printFun(1)**

are executed and it returns to

**printFun(2)**

and so on. In the output, values from 3 to 1 are printed and then 1 to 3 are printed. The memory stack has been shown in below diagram.



```
void printFun(3)
test=3
    1.  printf("%d",test);
    2.  printFun(2);
    3.  printf("i is %d",i);
    4.  return;

void printFun (2)
test=2
    1.  printf("i is %d",i);
    2.  printFun(1);
    3.  printf("i is %d",i);
    4.  return;

void printFun (1)
test=1
    1.  printf("i is %d",i);
    2.  printFun (0);
    3.  printf("i is %d",i);
    4.  return;

void printFun (0)
test=0
if(i<1)
return;
```

printFun(3) calls printFun(2)
printFun(2) calls printFun(1)
printFun(1) calls printFun(0)
Returns to printFun(3)
Returns to printFun(2)
Returns to printFun(1)

## Disadvantage of Recursion

: Note that both recursive and iterative programs have the same problem-solving powers, i.e., every recursive program can be written iteratively and vice versa. Recursive program has greater space requirements than iterative program as all functions will remain in the stack until the base case is reached. A recursive program also has greater time requirements because of function calls and return overhead.

## Advantages of Recursion

: Recursion provides a clean and simple way to write code. Some problems are inherently recursive like tree traversals, Tower of Hanoi, etc. For such problems, it is preferred to write recursive code. We can write such codes also iteratively with the help of the stack data structure.