# Recursion

A way of solving a problem by having a function calling itself.

## Properties of Recursion

1.Performing same operation each time with different inputs

2.In every step try to reduce the size of input to make the problem smaller

3.Base condition is required to stop recursion else infinite loop will occur

Example:-
```
def openGift(gift):
if gift == 1:
  print("This is the last gift and the number is: ",gift)
else:
   print("This gift number is: ",gift)
   return openGift(gift-1)
```

## Why do we need Recursion?

1. Recursion thinking is really important in programming and it helps us to reduce the big problems into smaller ones and easier to use
2. Recursive solution can be easy to read then iterative one

## When to choose recursion?

If you can divide a problem into sub problem and the sub problem should be same type

## How we know that sub problem is same type?

When you see a problem is beginning with following statement is maybe a good candidate of recursion

->If we can divide the problem into sub problem of same type.
->Write an algorithm to compute nth value.
->Implement a method to evaluate all

-->The main usage of recursion in data structure like trees and graph
-->Recursion is also used many algorithms like divide and conquer, greedy, dynamic programming etc

## How Recursion actually works?

Conditions:-

1. Function should call itself with smaller values
2. It should exit from infinite loop

Syntax:-

```
        def recursionFun(parameters):-
          if condition satisfied:'
            return sameValue
          else:
                return recursionFun(modifiedParameters)
```

```
def funOne():
  funTwo()
  print("In FunOne")
def funTwo():
  funThree()
  print("In funTwo")
def funThree():
  funFourth()
  print("In funThree")
def funFourth():
  print("In funFourth")
funOne()
```

# Recursive vs Iteration solution

1. All recursive call can be applied iteratively
2. Conditional statement decide the termination of recursion, while a control variable values decide the termination of iteration statement
3. No stack memory required incase of iteration, while incase of recursion system needs more time for pop and push element to stack memory which makes recursion less time efficient.

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

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

# When to use Recursion?

1. When we can break the problem into similar sub problem
2. When time and space complexity is not the constrain
3. when traverse a tree
4. recursion reduce the need of debug

# When avoid recursion?

1. If time and space complexity is a constrain
2. When we can break the problem into similar sub problem

# How to write recursion in 3 steps

Step 1. Recursive case ----> The flow

Step 2. Base case ----> The stopping criteria

Step 3. Unintentional case ----> The constraint

1. Write a program to Find the Factorial of a Number(n!)

   It is the product of all positive integer less than or equal to n

   0! = 1 and 1! = 1

```
def factorial(n):
    assert n>=0 and int(n) == n, "The number must be positive integer"
    if n in [0,1]:
        return 1
    else:
        return n*factorial(n-1)
```

# Programs

1. Write a program for nth Fibonacci number

Fibonacci sequence is a sequence of numbers in which each number is the sum of the two preceding ones and sequence start from 0 and 1.

Example :- 0 1 1 2 3 5 8 13 21 34 55 89 ….

2. Write a program to Find the sum of a positive integer number

Example :- 12345 == 1+2+3+4+5 = 15

# Programs

1. Write a program to calculate power of a Number

2. Write a program to find GCD(Greatest common divisor)

3. Write a program to convert a number from decimal to binary

5. Write a program to reverse a string

6. Write a program to check a string is palindrome