Recursion: -  
Recursion refers to a programming technique in which a function calls itself either directly or indirectly.  
Or  
Recursion is a technique for solving a large computational problem by repeatedly applying the same procedures to reduce it to successively smaller problems. A recursive procedure has two parts: one or more base case and a recursive step.  
  
Recursive function: - A function is said to be recursive function if it calls itself.

RECURSION IN PYTHON  
  
Recursive definition: -  
A recursive definition is a definition that is made in term of smaller version of itself.  
  
You can understand by following examples: -  
  
xn =x\*x\*x...\*x (Iterative definition or non-recursive)  
  
xn = x\*(x^n-1) for n>1 (recursive definition)  
       = x             for n = 1  
       = 1             for n = 0  
  
Condition to have recursive function: -  
• The Recursive case (or the inductive case)  
• The Base case (or the stopping case) - ALWAYS REQUIRED.

You can understand recursive function by following program: -

def compute(num):

if num == 1 :

return 1

else :

return (num + compute(num - 1))

number = 4

sum = compute(number)

print ("The sum of the series from 1...", number , "is" ,sum )

It solve like that :-

compute(4)

=4 + compute(3)

=4 + (3 + compute(2))

=4 + (3+ (2+compute(1)))

=4 + (3+ (2+1))

=4 + (3+3)

=4 + 6

=10  
  
 \* It is mandatory to have a base case to write a sensible recursion code.  
  
• If there is no base case or if the base case is never executed, an infinite recursion occurs.

Infinite recursion: - An Infinite recursion is when a recursive function calls itself endlessly.

• Infinite recursion can happen one of the following cases :-  
(i) Base Case Not Defined. When the recursive function has no BASE CASE defined, infinite recursion will occur.  
(ii)Base Case Not Reached. Sometimes you have written a base case but the condition to execute it, is never satisfied, hence the infinite recursion occurs.  
  
For example

def power (a , b):

if b == 0 :

return 1

else :

return a \* power(a , b - 1)

print ("Enter only positive number ! ")

num = int( input("Enter the base number :- "))

p = int (input("raised to the power of : "))

result = power (num , p)

print (num,"raised to the power of ", p , "is",result)

• The code of above program will face infinite recursion if you enter a negative value for power. In that way condition will never be true for a negative value of b.

Iterative version: -  
  
A recursive code may also be written in non-recursive way, which is the Iterative version of the same.  
  
For example: -

def power (a, b):

res = 1

if b == 0 :

return 1

else :

for i in range (b):

res = res \* a

return res

print ("Enter only positive number ! ")

num = int( input("Enter the base number :- "))

p = int (input("raised to the power of : "))

result = power (num , p)

print (num,"raised to the power of ", p , "is", result)

Binary search: -  
Binary search can work for only sorted arrays whereas linear search can work for both sorted as well as unsorted arrays.  
  
You can understand binary search method by following program: -

def bisearch (ar, key):

low = 0

high = len(ar)-1

while low <= high :

mid = int ((low+high)/2)

if key == ar[mid]:

return mid

elif key < ar [mid]:

high = mid - 1

else :

low = mid +1

else : # loop else

return -999

ar = [12,16,19,21,26,59,86,92,97]

item = int (input("Enter search item : " ))

res = bisearch(ar , item)

if res >= 0:

print (item , "FOUND at index ", res)

else :

print ("Sorry ! ",item ,"Not FOUND in array")

• In order to implement binary search on an array, following conditions must be fulfilled.  
(I)The given array or sequence must be sorted.  
(II) Its sort-order and size be known.  
  
• Binary search is very fast compared to linear search on one-dimensional arrays. However, linear search can work on both sorted and unsorted arrays while binary search can work with sorted arrays only. Also, binary search algorithm work with single-dimensional array where linear search algorithm can work with single and multi-dimensional arrays.

Recursive Binary Search: -  
Following program lists the recursive version of binary search algorithm.

def bisearch(ar, key , low , high):

if low > high :

return -999

mid = int ((low + high )/2)

if key == ar [mid]:

return mid

elif key < ar [mid]:

high = mid -1

return bisearch(ar, key , low , high)

else :

low = mid +1

return bisearch(ar, key , low , high)

ar = [12,16,19,21,26,59,86,92,97]

item = int (input("Enter search item : " ))

res = bisearch(ar , item,0,len(ar)-1)

if res >= 0:

print (item , "FOUND at index ", res)

else :

print ("Sorry ! ",item ,"Not FOUND in array")

RECURSION VS ITERATION: -  
  
• Recursion makes the code short and simple while iteration makes the code longer comparatively.  
• Recursion is slower than iteration due to overhead of multiple function calls and maintaining a stack for it.  
• In iteration, the code is executed repeatedly using the same memory space. That is, the memory space allocated once, is used for each pass of the loop.  
On the other hand in recursion, since it involves function call at each step, fresh memory is allocated for each recursive call. For this reason i.e., because of function call overhead, the recursive function runs slower than its Iterative counterpart.