RECURSION QUESTIONS

Table of Contents

LEVEL 0: Easy

1. Find factorial of number using recursion.
2. Find the nth term of Fibonacci series. 0,1,1,2,3,5,8…
3. Recursive function to reverse the string.
4. Print 1 to n numbers using recursion.
5. Print 1 to n numbers in reverse order (descending order).
6. Write recursive function to calculate sum of ranges of numbers
   Eg: sum of all numbers from a to b (a and b inclusive)
7. Find gcd of two numbers.
8. Recursive function to count occurrences of given character in string
9. Write a recursive function to find sum of all digits in a number.
   Eg. For number 101249 , sum of digits is 17
10. Find maximum number in the list of integers.
11. Recursive function to find power of number.
    Eg. Find num power of x, in most effective way

LEVEL 1: Moderate

1. Find ways to go to party.
   There are N persons who want to go to party. There is constraint that any person can either go alone or go in a pair. Calculate number of ways in which n persons can go to the party. Eg for n=3 , persons A,B,C can go as [A,B,C] , [A,(B,C)] , [(A,B),C] , [(A,C),B] = total 4 ways

2. Recursive function to check if string is palindrome.

3. Print this pattern.
   For n = 4 pattern is ****
   ***
   **
   *

4. Recursive function to merge two sorted lists.

5. Print a special series
   You are given a series defined by $T_n = (T_{n-2})^2 - (T_{n-1})$ .Here 1^{st} and 2^{nd} values are 0,1. Print the first N values.
   GFG question

6. Count of binary strings
   Count number of ways to make a binary string of length n, where there is no adjacent 1.

7. Print all subsets of array.

8. Print all n length binary strings not having adjacent 1's.

9. Print all permutations of a string.

10. Print all paths in 2D grid
    You are given a 2D grid (m × n), you are at the top left point and you need to reach bottom right point. You can only go right or down. Count all possible ways and also print all the paths.

11. Print all paths in 2D grid with diagonal move too
    You are given a 2D grid (m × n), you are at the top left point and you need to reach bottom right point. You can only go right or down and diagonally down too. Print all the paths.

12. Is subsequence
    Given two strings s and t, return true if s is a subsequence of t, or false otherwise. A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

LEVEL 2: Difficult

LEVEL 3: Expert

SOLUTIONS:

LEVEL 0:

1. Factorial of number

```
def fact(n):
    if n==1:            #base case
        return 1
    sub = fact(n-1)     #recursive subproblem
    return n*sub        #self work

print(fact(3))
```

2. Fibonacci Series

```
def fib(n):
    if(n<=1):           #base case
        return n
    sub1 = fib(n-1)     #recursive subproblem
    sub2 = fib(n-2)
    return sub1+sub2    #self work

print(fib(6))
```

3. Reverse strings

```
def reverse_string(s):
    if s=="" :                          #base case
        return ""
    return s[-1] + reverse_string(s[:-1])   #self work,recursive subproblem

print(reverse_string("python"))
```

4. Print 1 to n

```
def print_nums(n):
    if(n==0):           #base case
        return
    print_nums(n-1)     #recursive subproblem
    print(n)            #self work

print_nums(5)
```

5. Print 1 to n in descending order

```
def print_nums_desc(n):
    if(n==0):          #base case
        return
    print(n)           #self work
    print_nums_desc(n-1)    #recursive subproblem

print_nums_desc(5)
```

6. Range sum (sum of all numbers from start to end)

```
def range_sum(start,end):
    if start>end :                          #base case
        return 0
    return start + range_sum(start+1,end)   #self work,recursive subproblem

print(range_sum(3,5))
```

7. GCD(a,b) //here a>b. The Euclidean algorithm is a method for finding the greatest common divisor (GCD) of two numbers. It is based on the observation that if a and b are two positive integers with a > b, then the GCD of a and b is the same as the GCD of b and a % b, where % is the modulo operator.
   The Euclidean algorithm is very efficient, with a worst-case time complexity of O(log n), where n is the larger of the two numbers.

```
def gcd(a,b):
    if(a==0):           #base case
        return b
    sub = gcd(b%a,a)    #recursive subproblem
    return sub          #self work

print(gcd(42,18))
```

8. Character occurrences in string

```
def count_occurances(string,char):
    if len(string)==0:                          #base case
        return 0
    if string[0]==char:
        return 1 + count_occurances(string[1:],char)  #self work,recursive subproblem
    else:
        return count_occurances(string[1:],char)    #self work,recursive subproblem

print(count_occurances("python application",'p'))
```

9. Sum of digits of a number

```
def sum_digits(num):

    if num<10:                                  #base case
        return num
    return num%10 + sum_digits(num//10)         #self work , recursive case

print(sum_digits(101249))
```

10. Max number in array

```
def find_max(arr):
    if len(arr) == 1:
        return arr[0]
    else:
        return max(arr[0], find_max(arr[1:]))

if __name__ == '__main__':
    print(find_max([1, 2, 3, 9, 5, 6, 7, 8]))
```

11. Find N**X

```
def power(n,x):
    if x==1:
        return 1
    else:
        if(x%2):
            return n*n*n*power(n,x//2)
        else:
            return n*n*power(n,x//2)

print(power(2,4))
```

LEVEL 1:

1. Ways to go to party

   Let 3 persons are there A B C

   F(n) denotes total number of ways to go to party

   F(n) depends on 2 cases:

   If A goes alone: F(n) = number of ways B,C can go to party = F(n-1)

   If A goes in pair: F(n) = number of ways to make pair * (number of ways other n-2 can go to party)
   = (n-1) * F(n-2)

   So

   F(n) = F(n-1) + (n-1)*F(n-2)

```
def ways_to_party(n):
    if n<=1:
        return 1
    else:
        return ways_to_party(n-1) + (n-1)*ways_to_party(n-2)

print(ways_to_party(3))
```

2. Check Palindrome

```
def is_palindrome(s):
    if len(s) <= 1:
        return True
    else:
        return s[0] == s[-1] and is_palindrome(s[1:-1])

print(is_palindrome('abcba'))
```

3. Print Pattern

```
def pattern(n,j):
    if n==0:
       return
    if j==0:
        print("\r")
        pattern(n-1,n-1)
    else:
        print("*",end="")
        pattern(n,j-1)
pattern(4,4)
```

```
#Via Loop
n=4
for i in range(n,0,-1):
    for j in range(i):
        print("*",end="")
    print("\r")
```

4. Merge two sorted arrays

```
def merge_sorted(a,b,osf):
    if len(a)==0:
        return osf+b
    if len(b)==0:
        return osf+a
    else:
        if a[0]<b[0]:
            osf+=[a[0]]
            merge_sorted(a[1:],b,osf)
        else:
            osf+=[b[0]]
            merge_sorted(a,b[1:],osf)
    return osf

a= [2,4,5,7,10]
b= [1,3,6,8,9]
print(merge_sorted(a,b,[]))
```

5. GFG series

```
class Solution:
    def gfSeries(self, N : int) -> None:
        # code here
        ans = [0]*N
        if N>1:
            ans[0]=0
        if N>2:
            ans[1]=1

        def rec(N):
            if N<=2:
                return N-1
            else:
                temp = rec(N-2)
                temp2 =  temp*temp-rec(N-1)
                ans[N-1] = temp2
                return temp2
        rec(N)
```

6. Count of Binary strings

It forms a Fibonacci series, for n=1 count =2 (0,1) , for n=2 count = 3(00,01,10) , for n=3 count =5(000,100,010,001,101)

```
#count of binary strings of length n with no consecutive 1's
def count_binary_strings(n):
    if n<=2:
        return n+1
    else:
        return count_binary_strings(n-1)+count_binary_strings(n-2)

print(count_binary_strings(4))
```

7. All subsets

```
def subsets(i,n,array,s):     #s= output so far
    if i==n:
        print('['+s+']')
        return
    subsets(i+1,n,array,s+array[i])
    subsets(i+1,n,array,s)

array=["A","B","C"]
subsets(0,len(array),array,"")
```

8. Print binary strings

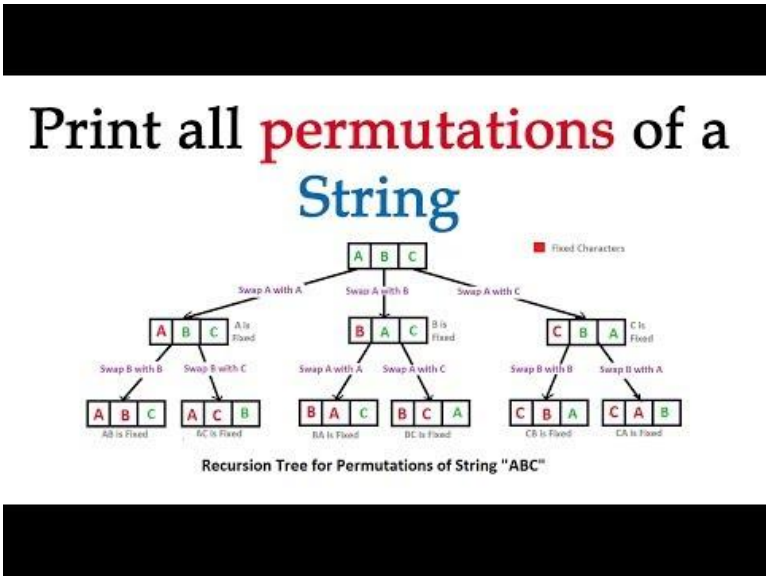
```
#osf = output so far
#one_flag = True if last value added was 1 else False
def binary_strings(i,n,osf,one_flag):
    if i==n:
        print(osf)
    else:
        if(one_flag):
            binary_strings(i+1,n,osf+"0",False)
        else:
            binary_strings(i+1,n,osf+"0",False)
            binary_strings(i+1,n,osf+"1",True)

if __name__ == '__main__':
    #n = length of required string
    n=int(input())
    binary_strings(0,n,"",False)
```

9. Print all permutations

```
def permutations(i,s):
    if i == len(s):
        print(''.join(s))
    else:
        for j in range(i,len(s)):
            s[i],s[j] = s[j],s[i]
            permutations(i+1,s)
            s[i],s[j] = s[j],s[i]

permutations(0,list('abc'))
```



10. All Paths

```
#To print all the paths,
#l = total length of path, for any path taken l = n-1+m-1
#i,j = current position m,n = destination
#s = string to store the path
def paths(i,j,m,n,s,l):
    if i==m:
        print(s+"D"*(l-len(s)))
        return 1
    elif j==n:
        print(s+"R"*(l-len(s)))
        return 1
    else:
        sub1 =  paths(i+1,j,m,n,s+"R",l)
        sub2 =  paths(i,j+1,m,n,s+"D",l)
        return sub1+sub2

n=2
m=3
print(paths(1,1,n,m,"",n-1+m-1))
```

To just get count of paths:

```
def count_paths(m,n):
    if m==1 or n==1:
        return 1
    return count_paths(m-1,n)+count_paths(m,n-1)

print(count_paths(2,3))
```

11. All paths including diagonal

```
#i,j = current position m,n = destination
#s = string to store the path
#R - right, B - bottom, D - diagonal
def paths(i,j,m,n,s):
    if i>m or j>n:
        return 0
    if i==m-1 and j==n-1:
        print(s)
        return 1

    else:
        sub1 =  paths(i+1,j,m,n,s+"R")
        sub2 =  paths(i,j+1,m,n,s+"B")
        sub3 =  paths(i+1,j+1,m,n,s+"D")
        return sub1+sub2+sub3

m=3
n=2
print(paths(0,0,m,n,""))
```

12. Is Subsequence

```
class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:
        def helper(s,t,i,j,m,n):
            if i==m: return True
            if j==n: return False
            if s[i]==t[j]: return helper(s,t,i+1,j+1,m,n)
            return helper(s,t,i,j+1,m,n)

        m=len(s)
        n=len(t)
        return helper(s,t,0,0,m,n)
```