**Question 1**

Given an integer n, return *true* if it is a power of three. Otherwise, return *false*.

An integer n is a power of three, if there exists an integer x such that n == 3x.

**Example 1:**

Input: n = 27

Output: true

Explanation: 27 = 33

**Example 2:**

Input: n = 0

Output: false

Explanation: There is no x where 3x = 0.

**Ans.** Solution from my leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def isPowerOfThree(self, n: int) -> bool:

        if n==1:

            return True

        elif n%3==1 or n<=0:

            return False

        else:

            return self.isPowerOfThree(n/3)

**Question 2**

You have a list arr of all integers in the range [1, n] sorted in a strictly increasing order. Apply the following algorithm on arr:

* Starting from left to right, remove the first number and every other number afterward until you reach the end of the list.
* Repeat the previous step again, but this time from right to left, remove the rightmost number and every other number from the remaining numbers.
* Keep repeating the steps again, alternating left to right and right to left, until a single number remains.

Given the integer n, return *the last number that remains in* arr.

**Example 1:**

Input: n = 9

Output: 6

Explanation:

arr = [1, 2,3, 4,5, 6,7, 8,9]

arr = [2,4, 6,8]

arr = [2, 6]

arr = [6]

**Ans.** Solution from my leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def lastRemaining(self, n: int) -> int:

        return 1 if n == 1 else 2 \* (1 + n // 2 - self.lastRemaining(n // 2))

**Question 3**

Given a set represented as a string, write a recursive code to print all subsets of it. The subsets can be printed in any order.

**Example 1:**

Input :  set = “abc”

Output : { “”, “a”, “b”, “c”, “ab”, “ac”, “bc”, “abc”}

**Example 2:**

Input : set = “abcd”

Output : { “”, “a” ,”ab” ,”abc” ,”abcd”, “abd” ,”ac” ,”acd”, “ad” ,”b”, “bc” ,”bcd” ,”bd” ,”c” ,”cd” ,”d” }

**Ans.** Solution from my leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def subsets(self, nums: List[int]) -> List[List[int]]:

        n, result = len(nums), []

        def powerSet(nums, i, subSet):

            if i==n:

                result.append(subSet)

                return

            powerSet(nums, i+1, subSet)

            powerSet(nums, i+1, subSet + [nums[i]])

        powerSet(nums, 0, [])

        return result

**Question 4**

Given a string calculate length of the string using recursion.

**Examples:**

Input : str = "abcd"

Output :4

Input : str = "GEEKSFORGEEKS"

Output :13

**Ans.**

def length(str):

if str == "":

return 0

return 1 + length(str[1:])

str = "iNeuron"

print("length of", str, "is", length(str)

**Question 5**

We are given a string S, we need to find count of all contiguous substrings starting and ending with same character.

**Examples :**

Input : S = "abcab"

Output : 7

There are 15 substrings of "abcab"

a, ab, abc, abca, abcab, b, bc, bca

bcab, c, ca, cab, a, ab, b

Out of the above substrings, there

are 7 substrings : a, abca, b, bcab,

c, a and b.

Input : S = "aba"

Output : 4

The substrings are a, b, a and aba

**Ans.**

**def** countSubstrs(str, i, j, n):

    # base cases

**if** (n **==** 1):

**return** 1

**if** (n <**=** 0):

**return** 0

    res **=** (countSubstrs(str, i **+** 1, j, n **-** 1)

**+** countSubstrs(str, i, j **-** 1, n **-** 1)

**-** countSubstrs(str, i **+** 1, j **-** 1, n **-** 2))

**if** (str[i] **==** str[j]):

        res **+=** 1

**return** res

**Question 6**

The [tower of Hanoi](https://en.wikipedia.org/wiki/Tower_of_Hanoi) is a famous puzzle where we have three rods and **N** disks. The objective of the puzzle is to move the entire stack to another rod. You are given the number of discs **N**. Initially, these discs are in the rod 1. You need to print all the steps of discs movement so that all the discs reach the 3rd rod. Also, you need to find the total moves.**Note:** The discs are arranged such that the **top disc is numbered 1** and the **bottom-most disc is numbered N**. Also, all the discs have **different sizes** and a bigger disc **cannot** be put on the top of a smaller disc. Refer the provided link to get a better clarity about the puzzle.

**Example 1:**

Input:

N = 2

Output:

move disk 1 from rod 1 to rod 2

move disk 2 from rod 1 to rod 3

move disk 1 from rod 2 to rod 3

3

Explanation:For N=2 , steps will be

as follows in the example and total

3 steps will be taken.

**Ans.**

def towerOfHanoi(n, source, destination, auxiliary):

if n == 1:

print("move disk 1 from rod", source, "to rod", destination)

return 1

moves = towerOfHanoi(n - 1, source, auxiliary, destination)

print("move disk", n, "from rod", source, "to rod", destination)

moves += towerOfHanoi(n - 1, auxiliary, destination, source)

return moves + 1

N = 2

total\_moves = towerOfHanoi(N, 1, 3, 2)

print("Total moves:", total\_moves)

**Question 7**

Given a string **str**, the task is to print all the permutations of **str**. A **permutation** is an arrangement of all or part of a set of objects, with regard to the order of the arrangement. For instance, the words ‘bat’ and ‘tab’ represents two distinct permutation (or arrangements) of a similar three letter word.

**Examples:**

Input: str = “cd”

**Output:** cd dc

**Input:** str = “abb”

**Output:** abb abb bab bba bab bba

**Ans.**

def swap(ch, i, j):

    temp = ch[i]

    ch[i] = ch[j]

    ch[j] = temp

def permutations(ch, curr\_index=0):

    if curr\_index == len(ch) - 1:

        print(''.join(ch))

    for i in range(curr\_index, len(ch)):

        swap(ch, curr\_index, i)

        permutations(ch, curr\_index + 1)

        swap(ch, curr\_index, i)

    s = 'ABC'

    permutations(list(s))

**Question 8**

Given a string, count total number of consonants in it. A consonant is an English alphabet character that is not vowel (a, e, i, o and u). Examples of constants are b, c, d, f, and g.

**Examples :**

Input : abc de

Output : 3

There are three consonants b, c and d.

Input : geeksforgeeks portal

Output : 12

**Ans.**

**def** isConsonant(ch):

    ch **=** ch.upper()

**return** **not** (ch **==** 'A' **or** ch **==** 'E' **or**

                ch **==** 'I' **or** ch **==** 'O' **or**

                ch **==** 'U') **and** ord(ch) >**=** 65 **and** ord(ch) <**=** 90

**def** totalConsonants(string, n):

**if** n **==** 1:

**return** isConsonant(string[0])

**return** totalConsonants(string, n **-** 1) **+** isConsonant(string[n**-**1])

string **=** "abc de"

print(totalConsonants(string, len(string)))