💡 \*\*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.

**def** isPowerOfThree(n):

**if** n **<=** 0:

**return** **False**

**while** n **%** 3 **==** 0:

n **//=** 3

**return** n **==** 1

*# Testing the function*

print(isPowerOfThree(27)) *# Output: True*

print(isPowerOfThree(0)) *# Output: False*

💡 \*\*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]

**Example 2:**

**def** lastRemaining(n):

left\_to\_right **=** **True**

remaining **=** n

step **=** 1

head **=** 1

**while** remaining **>** 1:

**if** left\_to\_right **or** remaining **%** 2 **==** 1:

head **+=** step

remaining **//=** 2

step **\*=** 2

left\_to\_right **=** **not** left\_to\_right

**return** head

*# Testing the function*

print(lastRemaining(9)) *# Output: 6*

#OUTPUT-

6

💡 \*\*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” }

**def** subsets(set\_str):

**def** generate\_subsets(set\_str, index, current\_set, subsets\_list):

subsets\_list**.**append(current\_set)

**for** i **in** range(index, len(set\_str)):

generate\_subsets(set\_str, i **+** 1, current\_set **+** set\_str[i], subsets\_list)

subsets\_list **=** []

generate\_subsets(set\_str, 0, "", subsets\_list)

**return** subsets\_list

*# Testing the function*

print(subsets("abc")) *# Output: ['', 'a', 'ab', 'abc', 'ac', 'b', 'bc', 'c']*

print(subsets("abcd")) *# Output: ['', 'a', 'ab', 'abc', 'abcd', 'abd', 'ac', 'acd', 'ad', 'b', 'bc', 'bcd', 'bd', 'c', 'cd', 'd']*

💡 \*\*Question 4\*\*

Given a string calculate length of the string using recursion.

**Examples:**

**def** calculateLength(string):

**if** string **==** '':

**return** 0

**return** 1 **+** calculateLength(string[1:])

*# Testing the function*

print(calculateLength("Hello")) *# Output: 5*

print(calculateLength("")) *# Output: 0*

#OUTPUT-

5

0

💡 \*\*Question 5\*\*

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

**Examples :**

**def** count\_contiguous\_substrings(s):

count **=** 0

**for** i **in** range(len(s)):

**for** j **in** range(i, len(s)):

**if** s[i] **==** s[j]:

count **+=** 1

**return** count

*# Example usage*

string **=** "abccba"

result **=** count\_contiguous\_substrings(string)

print(result)

#OUTPUT-

9

💡 \*\*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.

**Example 2:**

**def** tower\_of\_hanoi(n, source, destination, auxiliary):

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

print(f"Move disk 1 from rod {source} to rod {destination}")

**return** 1

**else**:

count **=** 0

count **+=** tower\_of\_hanoi(n **-** 1, source, auxiliary, destination)

print(f"Move disk {n} from rod {source} to rod {destination}")

count **+=** 1

count **+=** tower\_of\_hanoi(n **-** 1, auxiliary, destination, source)

**return** count

*# Example usage*

n **=** 2

total\_moves **=** tower\_of\_hanoi(n, 1, 3, 2)

print(total\_moves)

#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

💡 \*\*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

**import** itertools

**def** print\_permutations(string):

permutations **=** [''**.**join(p) **for** p **in** itertools**.**permutations(string)]

print(' '**.**join(permutations))

*# Example usage*

string **=** "cd"

print\_permutations(string)

#OUTPUT-

cd dc

💡 \*\*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 :**

**def** count\_consonants(string):

vowels **=** ['a', 'e', 'i', 'o', 'u']

count **=** 0

**for** char **in** string:

**if** char**.**isalpha() **and** char**.**lower() **not** **in** vowels:

count **+=** 1

**return** count

*# Example usage*

string **=** "Hello World"

consonant\_count **=** count\_consonants(string)

print(consonant\_count)

#OUTPUT-

7