**DSA – ASSIGNMENT 10**

💡 **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.

**Example 3:**

Input: n = -1

Output: false

Explanation: There is no x where 3x = (-1).

**Solution. :-**

* If n is less than or equal to 0, return false since a power of three must be a positive number.
* While n is divisible by 3, divide n by 3.
* After the loop, if n equals 1, return true; otherwise, return false.

**def isPowerOfThree(n):**

**if n <= 0:**

**return False**

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

**n /= 3**

**return n == 1**

**print(isPowerOfThree(27))**

💡 **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:**

Input: n = 1

Output: 1

**Solution. :-**

* Initialize a variable named leftToRight and set it to True. This variable will indicate the direction of traversal.
* Initialize two variables: remaining to keep track of the current remaining numbers and step to keep track of the current step size. Set remaining to n and step to 1.
* While remaining is greater than 1:
  + If leftToRight is True, or it's the first iteration, set start to 1 and end to remaining (inclusive).
  + If leftToRight is False, set start to remaining if it's odd, or remaining - 1 if it's even. Set end to 1 (inclusive).
  + Update step to step \* 2 since every second number is being removed.
  + Calculate the number of elements in the range using (end - start) // step + 1.
  + If remaining is odd or the number of elements is odd, update remaining to start + (step \* (number of elements - 1)).
  + Otherwise, update remaining to start + (step \* number of elements).
  + Toggle leftToRight by assigning it the logical NOT of its current value.
* Return remaining, which will be the last number that remains in arr.

**def lastRemaining(n):**

**leftToRight = True**

**remaining = n**

**step = 1**

**while remaining > 1:**

**if leftToRight or remaining % 2 == 1:**

**start = 1**

**end = remaining**

**else:**

**start = remaining if remaining % 2 == 1 else remaining - 1**

**end = 1**

**step \*= 2**

**numElements = (end - start) // step + 1**

**if remaining % 2 == 1 or numElements % 2 == 1:**

**remaining = start + (step \* (numElements - 1))**

**else:**

**remaining = start + (step \* numElements)**

**leftToRight = not leftToRight**

**return remaining**

**print(lastRemaining(9))**

💡 **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” }

**Solution. :-**

* Create an empty list to store the subsets.
* Define a recursive function generateSubsets that takes three parameters: the original string set, the current index, and the current subset current.
* In the recursive function:
  + Base case: If index is equal to the length of set, add the current subset to the list of subsets.
  + Recursive case:
    - Call the generateSubsets function recursively with the next index (index + 1) and the current subset current.
    - Append the current character of set at index to the current subset.
    - Call the generateSubsets function recursively with the next index and the updated current subset.
* Call the generateSubsets function with the initial index 0 and an empty string as the current subset.
* Return the list of subsets.

**def generateSubsets(set, index, current, subsets):**

**if index == len(set):**

**subsets.append(current)**

**return**

**generateSubsets(set, index + 1, current, subsets)**

**generateSubsets(set, index + 1, current + set[index], subsets)**

**def printSubsets(set):**

**subsets = []**

**generateSubsets(set, 0, '', subsets)**

**return subsets**

**print(printSubsets('abc'))**

💡 **Question 4** Given a string calculate length of the string using recursion.

**Example 1:**

Input : str = "abcd"

Output :4

**Example 2:**

Input : str = "GEEKSFORGEEKS"

Output :13

**Solution. :-**

* Define a recursive function calculateLength that takes a string str and an index as parameters.
* In the recursive function:
  + Base case: If index is equal to the length of the string, return 0.
  + Recursive case: Return 1 + calculateLength(str, index + 1), which adds 1 to the length of the substring starting from the next index.
* Call the calculateLength function with the string and an initial index of 0.
* Return the result of the recursive call.

**def calculateLength(str, index):**

**if index == len(str):**

**return 0**

**return 1 + calculateLength(str, index + 1)**

**# Test the function**

**str = "abcd"**

**print(calculateLength(str, 0))**

💡 **Question 5** We are given a string S, we need to find count of all contiguous substrings starting and ending with same character.

**Example 1 :**

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.

**Example 2:**

Input : S = "aba"

Output : 4

The substrings are a, b, a and aba

**Solution. :-**

* Initialize a variable count to 0 to keep track of the count of valid substrings.
* Iterate over each character ch in the string S:
  + For each character ch, initialize a variable freq to 0 to keep track of the frequency of the current character.
  + Iterate over each character c starting from the current character ch:
    - Increment freq by 1.
    - If the current character c is equal to the initial character ch, increment count by freq.
* Return the value of count, which represents the count of all contiguous substrings starting and ending with the same character.

**def countSubstrings(S):**

**count = 0**

**for i, ch in enumerate(S):**

**freq = 0**

**for c in S[i:]:**

**freq += 1**

**if c == ch:**

**count += freq**

**return count**

**# Test the function**

**S = "abcab"**

**print(countSubstrings(S))**

💡 **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:**

Input:

N = 3

Output:

move disk 1 from rod 1 to rod 3

move disk 2 from rod 1 to rod 2

move disk 1 from rod 3 to rod 2

move disk 3 from rod 1 to rod 3

move disk 1 from rod 2 to rod 1

move disk 2 from rod 2 to rod 3

move disk 1 from rod 1 to rod 3

7

Explanation: For N=3 , steps will be as follows in the example and total 7 steps will be taken.

**Solution. :-**

* Define a recursive function towerOfHanoi that takes five parameters: the number of discs N, the source rod src, the auxiliary rod aux, the destination rod dest, and a counter variable count to keep track of the total moves.
* Base case: If N is equal to 1, print the step to move the disc from the source rod to the destination rod and increment the count by 1.
* Recursive case:
  + Move N-1 discs from the source rod to the auxiliary rod using the destination rod as the auxiliary rod by calling the towerOfHanoi function recursively with N-1, src, dest, and aux as parameters.
  + Print the step to move the Nth disc from the source rod to the destination rod and increment the count by 1.
  + Move the N-1 discs from the auxiliary rod to the destination rod using the source rod as the auxiliary rod by calling the towerOfHanoi function recursively with N-1, aux, src, and dest as parameters.
* Call the towerOfHanoi function with the initial parameters: N, source rod index (e.g., 1), auxiliary rod index (e.g., 2), destination rod index (e.g., 3), and a counter variable initialized to 0.
* Print the value of the counter variable, which represents the total number of moves.

**def towerOfHanoi(N, src, aux, dest, count):**

**if N == 1:**

**print("move disk 1 from rod", src, "to rod", dest)**

**count += 1**

**return count**

**count = towerOfHanoi(N - 1, src, dest, aux, count)**

**print("move disk", N, "from rod", src, "to rod", dest)**

**count += 1**

**count = towerOfHanoi(N - 1, aux, src, dest, count)**

**return count**

**# Test the function**

**N = 2**

**totalMoves = towerOfHanoi(N, 1, 2, 3, 0)**

**print(totalMoves)**

💡 **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.

**Example 1:**

Input: str = “cd”

Output: cd dc

**Example 2:**

Input: str = “abb”

Output: abb abb bab bba bab bba

**Solution. :-**

* Define a recursive function permutations that takes three parameters: the original string str, the current index start, and an empty string current.
* In the recursive function:
  + Base case: If start is equal to the length of str, print the current permutation.
  + Recursive case:
    - Iterate over each character ch in str starting from the start index:
      1. Swap the characters at indices start and i.
      2. Call the permutations function recursively with the next index (start + 1) and the updated current string.
      3. Swap the characters back to their original positions to restore the original order.
* Call the permutations function with the initial index 0 and an empty string as the current permutation.

**def permutations(str, start, current):**

**if start == len(str):**

**print(current)**

**return**

**for i in range(start, len(str)):**

**strList = list(str)**

**strList[start], strList[i] = strList[i], strList[start]**

**newStr = "".join(strList)**

**permutations(newStr, start + 1, current + newStr[start])**

**strList[start], strList[i] = strList[i], strList[start] # Swap back to restore original order**

**# Test the function**

**str = "cd"**

**permutations(str, 0, "")**

💡 **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.

**Example 1 :**

Input : abc de

Output : 3

There are three consonants b, c and d.

**Example 2:**

Input : geeksforgeeks portal

Output : 12

**Solution. :-**

* Initialize a variable count to 0 to keep track of the number of consonants.
* Convert the given string to lowercase to handle both uppercase and lowercase consonants uniformly.
* Iterate over each character ch in the string:
  + Check if ch is an alphabet character and not a vowel (a, e, i, o, u).
  + If it is a consonant, increment the count by 1.
* Return the value of count, which represents the total number of consonants in the string.

**def countConsonants(string):**

**count = 0**

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

**string = string.lower()**

**for ch in string:**

**if ch.isalpha() and ch not in vowels:**

**count += 1**

**return count**

**# Test the function**

**string = "abc de"**

**print(countConsonants(string))**