ASSIGNMENT 10

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`.

```
def is_power_of_three(n):
   if n == 1:
      return True
   elif n < 1 or n % 3 != 0:
      return False
   else:
      return is_power_of_three(n // 3)
```

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`.

```
 def last_remaining_number(n):
   arr = list(range(1, n + 1))
   return recursive_remove(arr)

def recursive_remove(arr):
   if len(arr) == 1:
      return arr[0]

   # Remove every other number from left to right
   arr = arr[1::2]

   # Reverse the remaining list and remove every other number from right to left
   arr = arr[::-1][1::2][::-1]

   return recursive_remove(arr)
```

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.

```
def generate_subsets(s, curr_subset, index):
    if index == len(s):
        print(curr_subset)
        return

    # Exclude current character
    generate_subsets(s, curr_subset, index + 1)

    # Include current character
    generate_subsets(s, curr_subset + s[index], index + 1)

def print_subsets(s):
    generate_subsets(s, "", 0)
```

4. Given a string calculate length of the string using recursion.

```
def calculate_length(s):
    if s == "":
        return 0
    else:
        return 1 + calculate_length(s[1:])
```

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

```
def count_contiguous_substrings(S):
    if len(S) <= 1:
        return 0

    count = count_contiguous_substrings(S[1:])

    if S[0] == S[-1]:
        count += 1

    return count
```

6. The 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.

```
def tower_of_hanoi(n, source, destination, auxiliary):
    if n == 1:
        print(f"Move disc 1 from {source} to {destination}")
        return 1

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

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

    moves += 1

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

    return moves
```

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.

```
def generate_permutations(s, l, r):
    if l == r:
        print(''.join(s))
    else:
        for i in range(l, r + 1):
            s[l], s[i] = s[i], s[l]  # Swap characters
            generate_permutations(s, l + 1, r)  # Recursive call
            s[l], s[i] = s[i], s[l]  # Backtrack (undo the swap)

def print_permutations(str):
    n = len(str)
    generate_permutations(list(str), 0, n - 1)
```

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.

```
def count_consonants(s):
    if s == "":
        return 0
    elif s[0].lower() in "aeiou":
        return count_consonants(s[1:])
    else:
        return 1 + count_consonants(s[1:])
```