Recursion-3

In this module, we are going to solve a few advanced problems using recursion.

Problem Statement: Return Subsequences

Given a string (let's say of length n), return all the subsequences of the given string. **Subsequences** contain all the strings of length varying from 0 to n. But the order of characters should remain the same as in the input string.
Note: The order of subsequences is not important.

Approach:

For example, If we have a string : **"abc"**, Then its subsequences are :- **"a", "b", "c", "ab", "ac", "bc", "abc"**

- First, we select the first character i.e. 'a', add it to a list, and fix it. ----> **a**
- Take next char i.e. 'b', add it to a list and fix it ----> **ab**
- Keep taking next chars until string ends ---->**abc**
- Delete up to first char, start taking char from second
- position from first char till string ends ----> **ac**
- Similarly, go for rest chars ---->**b, bc, c**
- If we have a string of N length. Then the number of its non-empty subsequences is (**2^n-1**). In the end, we return the list of subsequences.

Note: The code written from the above insights can be accessed in the solution tab in the question itself.

Problem Statement: Print All Subsequences

Given a string (let's say of length n), print all the subsequences of the given string.
Subsequences contain all the strings of length varying from 0 to n. But the order of characters should remain the same as in the input string.

Approach: This problem is fairly simple and similar in approach to the above problem, with the only change being that instead of adding the subsequences to a list, we directly print them.

Problem Statement: Print Keypad Combinations

Given an integer n, using the phone keypad find out and print all the possible strings that can be made using digits of input n.
Note: The order of strings is not important. Just print different strings in new lines.

The phone keypad looks like this:

Approach:
It can be observed that each digit can represent 3 to 4 different alphabets (apart from 0 and 1). So the idea is to form a recursive function. You can follow the given steps:

- Map the digit with its string of probable alphabets, i.e 2 with "abc", 3 with "def" etc. (i.e. create a database of this mapping)

- Create a recursive function that takes the following parameters: output string, number array, current index, and length of number array.
- **Base Case:** If the current index is equal to the length of the number array then print the output string.
- Extract the string at **digit[current_index]** from the Map, where the digit is the input number array.
- Run a loop to traverse the string from **start** to **end.**
- For every index again call the recursive function with the output string concatenated with the **i^{th}** character of the string and the **current_index + 1**.

Note: The code written from the above insights can be accessed in the solution tab in the question itself.