C1: Dùng Bit Serial Mapping (Lập bản đồ bit nhị phân):

Ý tưởng:

Số tổ hợp n được tạo ra từ một dãy số bất kì có thể mapping với dãy bit nhị phân từ 0 → n-1.

Lấy ví dụ một list có 3 phần tử: lst = [1,2,3]. → n = 2^3 = 8 (Size của lst = 3)

Xét dãy bit từ 0 → n-1:

0 = 0b 000 ⇔ tập rỗng = [ ]
1 = 0b 00**1** ⇔ tập chứa phần tử thứ 1 = [1]
2 = 0b 0**1**0 ⇔ tập chứa phần tử thứ 2 = [2]
3 = 0b 0**11** ⇔ tập chứa phần tử thứ 1 + 2 = [1, 2]
4 = 0b **1**00 ⇔ tập chứa phần tử thứ 3 = [3]
5 = 0b **1**0**1** ⇔ tập chứa phần tử thứ 1 + 3 = [1, 3]
6 = 0b **11**0 ⇔ tập chứa phần tử thứ 2 + 3 = [2, 3]
7 = 0b **111** ⇔ tập chứa phần tử thứ 1 + 2 + 3 = [1, 2 , 3]

C2: Dùng Element Adding

Ý tưởng:
Xây dựng tất cả các tổ hợp từ tập rỗng, chèn thêm một phần tử mới cho các tập trước đó sau mỗi lần lặp.

Lấy ví dụ một list có 3 phần tử: lst = [1,2,3].

Khởi tạo giá trị cho biến solution là một tập rỗng = [**[ ]**]

Lần lặp đầu tiên, thêm phần tử thứ [**1**] vào biến solution:
solution = [[ ], [**1**]]

Lần lặp thứ hai:
solution = [[ ], [1], [**2**], [1, **2**]]

Lần lặp thứ ba:
solution = [[ ], [1], [2], [1, 2], [**3**], [1, **3**], [2, **3**], [1, 2, **3**]]

Sau 3 lần lặp:

All subsets = [[ ], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]

C3: Dùng hàm <combinations> có sẵn trong Python

itertools.combinations(n , k) → Tìm ra số tổ hợp theo C_n^k

Ví dụ:
combinations('ABCD', 2) --> AB AC AD BC BD CD

Lấy ví dụ một list có 3 phần tử: lst = [1,2,3].

Tập rỗng = [ ] ⇔ C_3^0

Tập chứa 1 phần tử = [1], [2], [3] ⇔ C_3^1

Tập chứa 2 phần tử = [1, 2], [1, 3], [2,3] ⇔ C_3^2

Tập chứa 3 phần tử = [1,2,3] ⇔ C_3^3