In [2]:
```
def combinations(candidates):
    curr = []
    pos = 0
    util(candidates, pos, curr)


def util(candidates, pos, curr):
    if pos == len(candidates):
        print(curr)
        return

    curr.append(candidates[pos])
    util(candidates, pos + 1, curr)
    curr.pop()

    util(candidates, pos + 1, curr)

combinations([1,2,3])

```

```
[1, 2, 3]
[1, 2]
[1, 3]
[1]
[2, 3]
[2]
[3]
[]
```

In [6]:
```
def combinations(candidates):
    curr = []
    pos = 0
    util(candidates, pos, curr)

def util(candidates, pos, curr):
    if pos == len(candidates):
        print(curr)
        return

    util(candidates, pos + 1, curr + [candidates[pos] ] )
    util(candidates, pos + 1, curr)

combinations([1,2,3])
```

```
[1, 2, 3]
[1, 2]
[1, 3]
[1]
[2, 3]
[2]
[3]
[]
```

In [4]:
```
a = []
print(a + [1])

a = [1,2]
print(a + [3])
```

```
[1]
[1, 2, 3]
```

In [ ]:
```

```

In [9]:
```
def combinations(candidates, target_sum):
    curr = []
    pos = 0
    util(candidates, pos, curr, target_sum)


def util(candidates, pos, curr, target_sum):
    if target_sum == 0:
        print(curr)
        return

    if target_sum < 0:
        return

    if pos == len(candidates):
        return

    ## modify this part
    curr.append(candidates[pos])
    util(candidates, pos, curr, target_sum-candidates[pos])
    curr.pop()

    util(candidates, pos + 1, curr, target_sum)

combinations([1,2,3], 4)
```

```
[1, 1, 1, 1]
[1, 1, 2]
[1, 3]
[2, 2]
```

In [ ]:
```
def combinations(candidates):
    curr = []
    pos = 0
    util(candidates, pos, curr, sum)


def util(candidates, pos, curr):
    if sum <= 0:
        if sum == 0:
            print(curr)
        return

    curr.append(candidates[pos])
    util(candidates, pos, curr, sum - candidates[pos])
    curr.pop()

    util(candidates, pos + 1, curr)
```

In [ ]:
```
class Solution(object):
    def combinationSum(self, candidates, target):
        result = []
        curr = []
        pos = 0
        self.util(candidates, pos, curr, target, result)
        return result
    def util(self, candidates, pos, curr, total, result):
        if total <= 0 or pos >= len(candidates):
            if total == 0:
                result.append(curr[:])
            return

        curr.append(candidates[pos])
        self.util(candidates, pos, curr, total - candidates[pos], result)
        curr.pop()
        self.util(candidates, pos + 1, curr, total, result)
```

In [ ]:
```java
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {

        List<List<Integer>> list = new ArrayList<List<Integer>>();

          backTrack(list, new ArrayList<Integer>(), candidates, target, 0);

          return list;
    }

    public void backTrack(List<List<Integer>> list, ArrayList<Integer> tempLis

        if(remain < 0) return;

        else if(remain == 0){
            list.add(new ArrayList<Integer>(tempList));
        }

        else{
            for(int i=start; i<can.length; i++){

                tempList.add(can[i]);

                backTrack(list, tempList, can, remain-can[i], i);
                tempList.remove(tempList.size()-1);
            }
        }

    }

}
```

In [ ]:
```
```

https://leetcode.com/problems/generate-parentheses/

In [ ]:
```python
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        res = []

        def backtrack(s, left, right):
            if len(s) == 2*n:
                res.append(s)
                return
            if left < n:
                backtrack(s+ '(', left+1, right)
            if right < left:
                backtrack(s+ ')', left, right+1)

        backtrack('',0 , 0)
        return res
```

In [ ]:
```
class Solution {
    public List<String> generateParenthesis(int n) {
        List<String> res = new ArrayList<String>();
        recurse(res, 0, 0, "", n);
        return res;
    }

    public void recurse(List<String> res, int left, int right, String s, int n
        if (s.length() == n * 2) {
            res.add(s);
            return;
        }

        if (left < n) {
            recurse(res, left + 1, right, s + "(", n);
        }

        if (right < left) {
            recurse(res, left, right + 1, s + ")", n);
        }
    }
}
```

In [ ]:
```
class Solution {
    public List<String> generateParenthesis(int n) {
        List<String> ans = new ArrayList<>();
        gen(n, n, ans, new StringBuilder());
        return ans;
    }

    private void gen(int open , int close, List<String> ans, StringBuilder sb)
        if(open  0 && close  0){
            ans.add(sb.toString());
            return;
        }
        if(open == 0){
            sb.append(")");
            gen(open, close-1, ans, sb);
            sb.deleteCharAt(sb.length()-1);
            return;
        }
        if(open == close){
            sb.append("(");
            gen(open-1, close, ans, sb);
            sb.deleteCharAt(sb.length()-1);
            return;
        }
        sb.append("(");
        gen(open - 1, close, ans, sb);
        sb.deleteCharAt(sb.length()-1);
        sb.append(")");
        gen(open, close - 1, ans, sb);
        sb.deleteCharAt(sb.length()-1);
        return;
    }
}
```