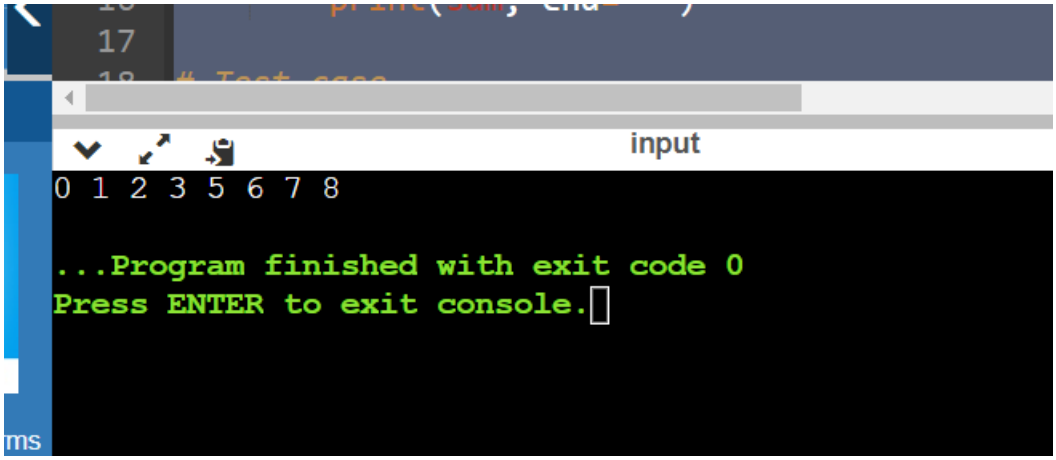
Day – 9 Recursion

Problem Statement: Given an array print all the sum of the subset generated from it, in the increasing order.

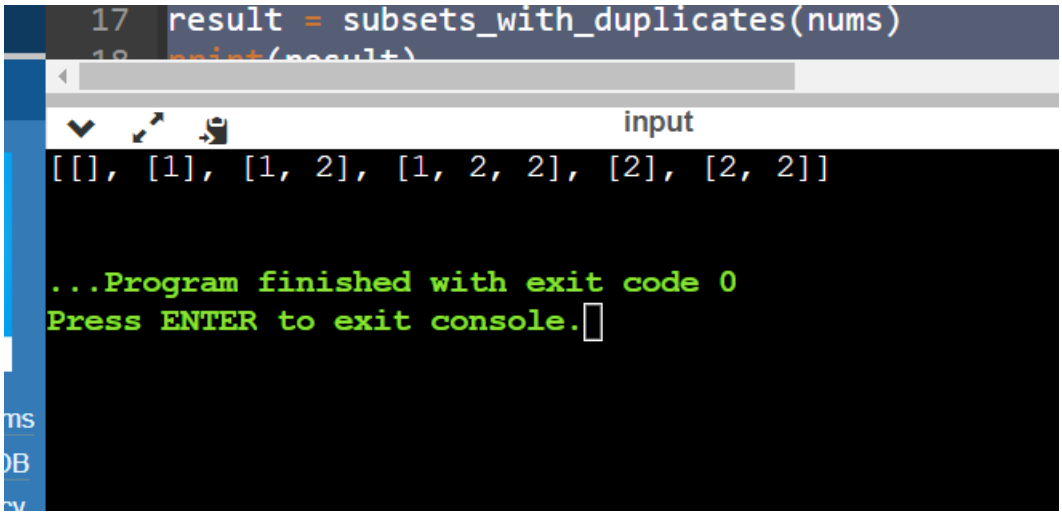
```
def print_subset_sums(arr):
    n = len(arr)
    subset_sums = []

    def generate_subsets(curr_sum, idx):
        if idx == n:
            subset_sums.append(curr_sum)
            return

        generate_subsets(curr_sum + arr[idx], idx + 1)
        generate_subsets(curr_sum, idx + 1)

    generate_subsets(0, 0)
    subset_sums.sort()
    for sum in subset_sums:
        print(sum, end=" ")

# Test case
arr = [5, 2, 1]
```

print_subset_sums(arr)



Problem Statement: Given an array of integers that **may contain duplicates** the task is to return all possible subsets. Return only **unique subsets** and they can be in any order

```
def subsets_with_duplicates(nums):
    nums.sort()
    subsets = []
    generate_subsets(nums, 0, [], subsets)
    return subsets


def generate_subsets(nums, index, current, subsets):
    subsets.append(current[:])
    for i in range(index, len(nums)):
        if i > index and nums[i] == nums[i - 1]:
            continue
        current.append(nums[i])
        generate_subsets(nums, i + 1, current, subsets)  # Generate subsets recursively
        current.pop()


nums = [1, 2, 2]
result = subsets_with_duplicates(nums)
print(result)
```



Problem Statement:

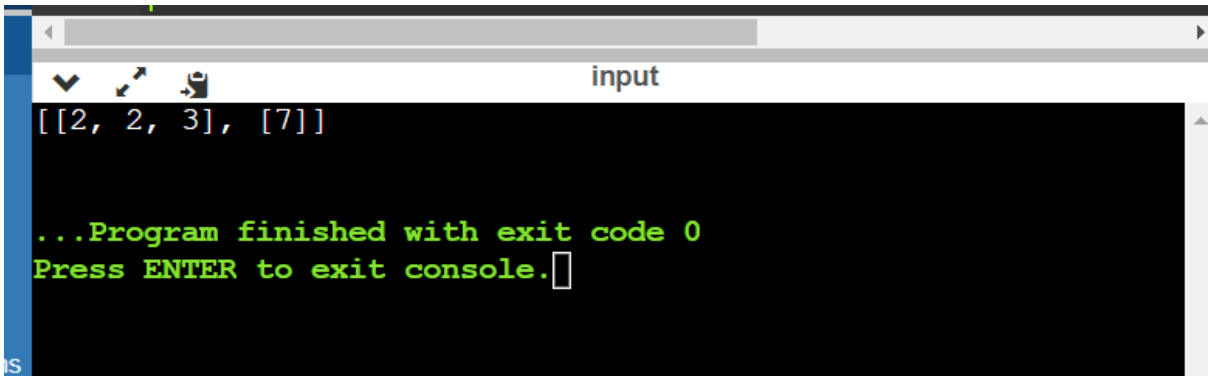
Given an array of distinct integers and a **target**, you have to return *the list of all unique combinations where the chosen numbers sum to* target. You may return the combinations in any order.

The same number may be chosen from the given array an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is guaranteed that the number of unique combinations that sum up to **target** is less than **150** combinations for the given input.

```
def combinationSum(candidates, target):
    results = []
    backtrack(candidates, target, [], results)
    return results


def backtrack(candidates, target, combination, results):
    if target < 0:
        return
    if target == 0:
        results.append(combination)
```

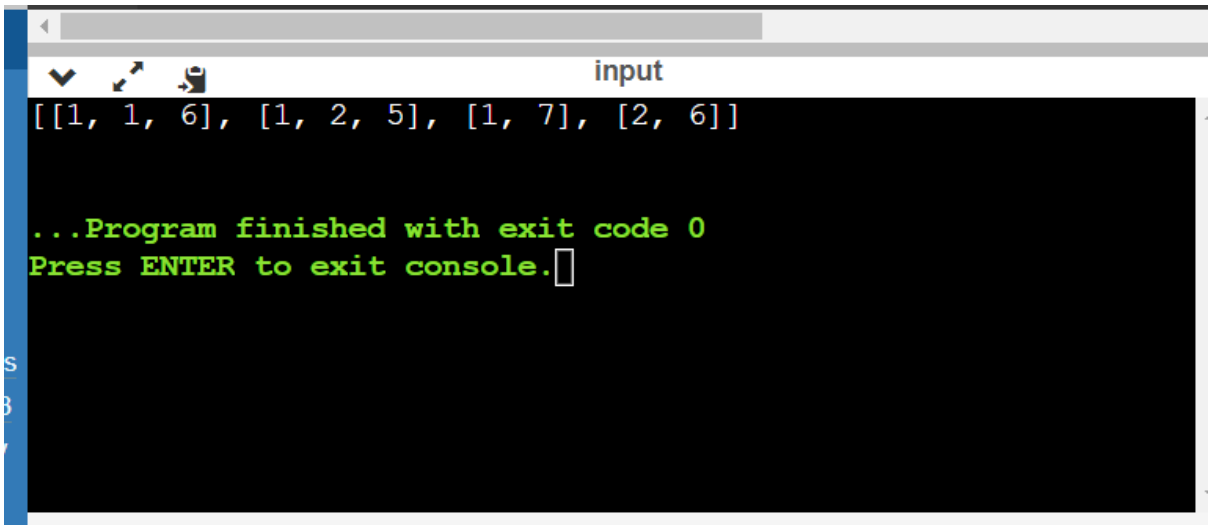
```
        return
    for i in range(len(candidates)):
        num = candidates[i]
        backtrack(candidates[i:], target - num, combination + [num], results)
array = [2, 3, 6, 7]
target = 7
result = combinationSum(array, target)
print(result)
```



Problem Statement: Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target. Each number in candidates may only be used once in the combination.

```
def combinationSum2(candidates, target):
    candidates.sort()
    result = []

    def backtrack(combination, remaining, start):
        if remaining == 0:
            result.append(combination)
            return
        if remaining < 0 or start == len(candidates):
            return
```

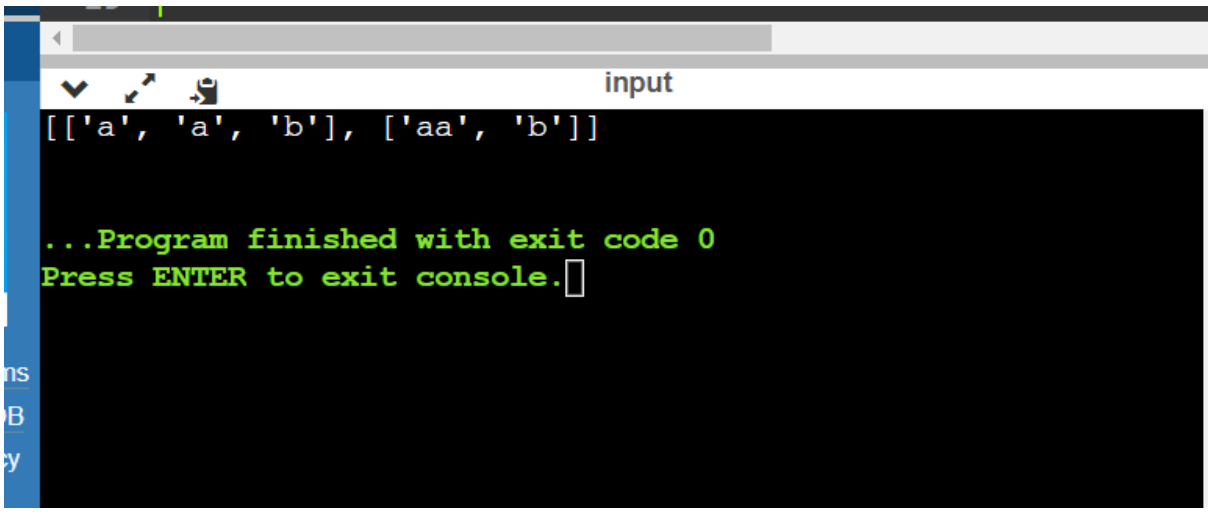
```
        for i in range(start, len(candidates)):
            if i > start and candidates[i] == candidates[i - 1]:
                continue

            candidate = candidates[i]
            if candidate > remaining:
                break
            backtrack(combination + [candidate], remaining - candidate, i + 1)

    backtrack([], target, 0)
    return result
candidates = [10, 1, 2, 7, 6, 1, 5]
target = 8
result = combinationSum2(candidates, target)
print(result)
```



Problem Statement: You are given a string s, partition it in such a way that every substring is a palindrome. Return all such palindromic partitions of s.

```
def is_palindrome(string):
    return string == string[::-1]
```

```
def partition_palindrome(s):
    result = []
    current_partition = []

    def backtrack(start):
        if start >= len(s):
            result.append(current_partition[:])
            return

        for end in range(start, len(s)):
            substring = s[start:end+1]
            if is_palindrome(substring):
                current_partition.append(substring)
                backtrack(end+1)
                current_partition.pop()

    backtrack(0)
    return result
s = "aab"
result = partition_palindrome(s)
print(result)
```



Problem Statement: Given **N** and **K**, where N is the sequence of numbers from **1 to N([1,2,3..... N])** find the **Kth permutation sequence**.

```
def getPermutation(N, K):
    numbers = list(range(1, N+1))
    result = ''

    def findPermutation(n, k):
        nonlocal numbers, result
        if n == 1:
            result += str(numbers[0])
            return

        factorial = 1
        for i in range(2, n):
            factorial *= i

        index = (k - 1) // factorial
        result += str(numbers[index])
        numbers.pop(index)
        k -= index * factorial

        findPermutation(n - 1, k)
```

```
    findPermutation(N, K)

    return result
```

```
N = 3

K = 3

permutation = getPermutation(N, K)

print(permutation)
```