Question 1

Convert 1D Array Into 2D Array

You are given a **0-indexed** 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with m rows and n columns using **all** the elements from original.

The elements from indices 0 to n - 1 (**inclusive**) of original should form the first row of the constructed 2D array, the elements from indices n to 2 * n - 1 (**inclusive**) should form the second row of the constructed 2D array, and so on.

Return *an* m x n *2D array constructed according to the above procedure, or an empty 2D array if it is impossible*.

```
def convertTo2DArray(original, m, n):

    if len(original) != m * n:

        return []

    result = [[0] * n for _ in range(m)]

    for i in range(m * n):

        row = i // n

        col = i % n

        result[row][col] = original[i]

    return result
```

Question 2

You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase **may be** incomplete.

Given the integer n, return *the number of* ***complete rows*** *of the staircase you will build*.

```
def buildStaircase(n):
    completeRows = 0
    coinsRequired = 1
    while n >= coinsRequired:
        completeRows += 1
        n -= coinsRequired
        coinsRequired += 1
    return completeRows
```

Question 3

Given an integer array nums sorted in **non-decreasing** order, return *an array of **the squares of each number** sorted in non-decreasing order*

```
def sortedSquares(nums):
    left = 0
    right = len(nums) - 1
    squares = []
    while left <= right:
        if abs(nums[left]) >= abs(nums[right]):
            squares.append(nums[left] ** 2)
            left += 1
        else:
            squares.append(nums[right] ** 2)
            right -= 1
    return squares[::-1]
```

Question 4

Given two **0-indexed** integer arrays nums1 and nums2, return *a list* answer *of size* 2 *where:*

- answer[0] *is a list of all* ***distinct*** *integers in* nums1 *which are* ***not*** *present in* nums2*.*
- answer[1] *is a list of all* ***distinct*** *integers in* nums2 *which are* ***not*** *present in* nums1.

Note that the integers in the lists may be returned in **any** order

```
def findDisjoint(nums1, nums2):

    set1 = set(nums1)

    set2 = set(nums2)

    distinct_nums1 = list(set1 - set2)

    distinct_nums2 = list(set2 - set1)

    return [distinct_nums1, distinct_nums2]
```

Question 5

Given two integer arrays arr1 and arr2, and the integer d, *return the distance value between the two arrays*.

The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

```
def findDistanceValue(arr1, arr2, d):

    count = 0

    for num1 in arr1:

        valid = True

        for num2 in arr2:

            if abs(num1 - num2) <= d:

                valid = False

                break

        if valid:

            count += 1

    return count
```

Question 6

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **once** or **twice**, return *an array of all the integers that appears **twice***.

You must write an algorithm that runs in O(n) time and uses only constant extra space.

```
def findDuplicates(nums):
    result = []
    for num in nums:
        index = abs(num) - 1
        if nums[index] < 0:
            result.append(abs(num))
        else:
            nums[index] *= -1
    return result
```

Question 7

Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

- [4,5,6,7,0,1,2] if it was rotated 4 times.
- [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that **rotating** an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in O(log n) time.

```
def findMin(nums):
    left = 0
    right = len(nums) - 1
    while left < right:
        mid = (left + right) // 2
        if nums[mid] > nums[right]:
            left = mid + 1
        else:
            right = mid
    return nums[left]
```

Question 8

An integer array original is transformed into a **doubled** array changed by appending **twice the value** of every element in original, and then randomly **shuffling** the resulting array.

Given an array changed, return original *if* changed *is a* ***doubled*** *array. If* changed *is not a* ***doubled*** *array, return an empty array. The elements in* original *may be returned in* ***any*** *order*.

```
from collections import defaultdict

def findOriginalArray(changed):
    count = defaultdict(int)
    for num in changed:
        count[num] += 1
    for num, freq in count.items():
        if freq % 2 != 0:
            return []
    original = []
    for num in changed:
        if count[num] > 0:
            count[num] -= 1
            original.append(num // 2)
    return original
```