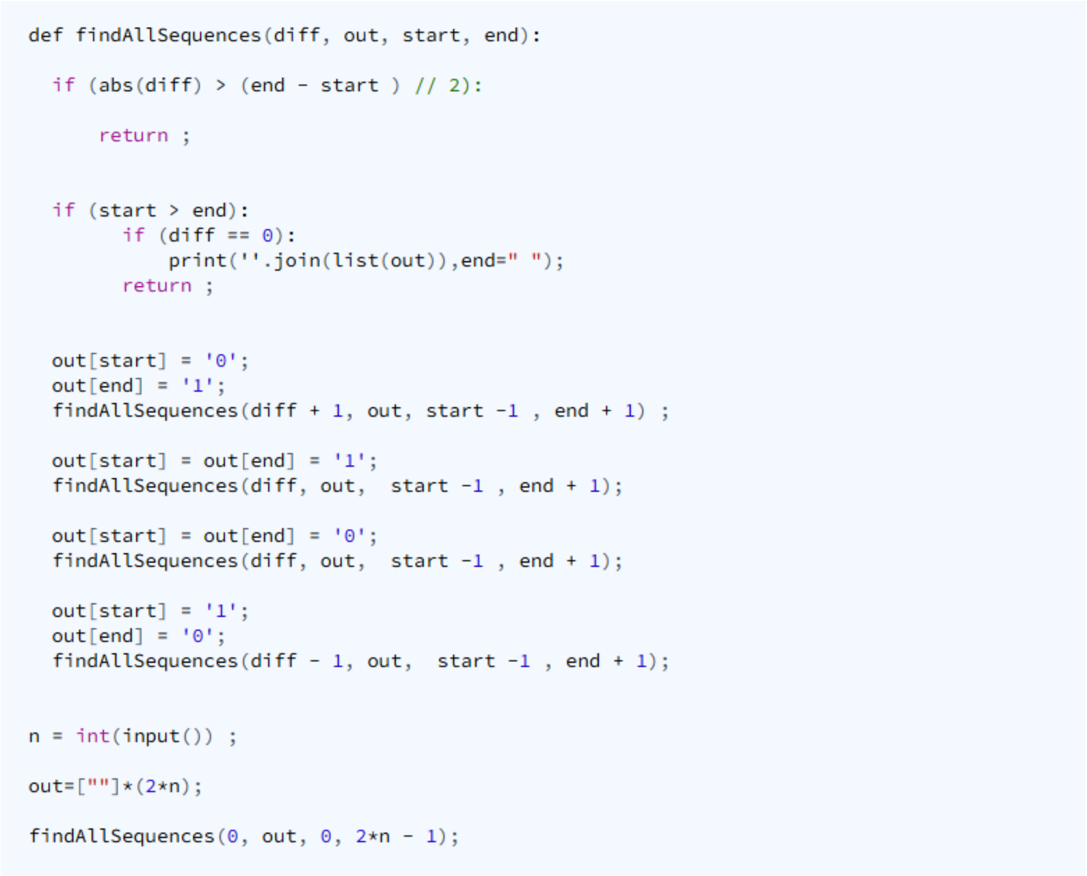
**Instruction:**

- Define **functions** for as many cases as possible for each program. AT LEAST one function for each program is a must.
- For each program, you must define a function, **_test()**, with assert statements that check the correctness of each function. TAs will evaluate if you have used any test function with assert or not, and accordingly assign a score. It will be counted towards final evaluation.

1. Given a number n, find all binary sequences of length 2n such that sum of first n bits is same as sum of last n bits.
Examples:
Input: N = 2
Output:
0101 1111 1001 0110 0000 1010
Input: N = 3
Output:
011011 001001 011101 010001 101011 111111
110011 101101 100001 110101 001010 011110
010010 001100 000000 010100 101110 100010
110110 100100

The following code segment is given to you to solve this question. However, it has many bugs. Debug the code:



```
def findAllSequences(diff, out, start, end):

  if (abs(diff) > (end - start ) // 2):

      return ;


  if (start > end):
        if (diff == 0):
            print(''.join(list(out)),end=" ");
        return ;


  out[start] = '0';
  out[end] = '1';
  findAllSequences(diff + 1, out, start -1 , end + 1) ;

  out[start] = out[end] = '1';
  findAllSequences(diff, out,  start -1 , end + 1);

  out[start] = out[end] = '0';
  findAllSequences(diff, out,  start -1 , end + 1);

  out[start] = '1';
  out[end] = '0';
  findAllSequences(diff - 1, out,  start -1 , end + 1);


n = int(input()) ;

out=[""]*(2*n);

findAllSequences(0, out, 0, 2*n - 1);
```

Hint - There are 5 potential errors(counting repeated mistakes) .

**Input Format**

First line of the input contains the number n .

**Constraints**

1 <= n <= 10

**Output Format**

Output space separated binary sequences of length 2n such that sum of first n bits is same as sum of last n bits. (if you follow the given code the output will always match , this ensures you make corrections in this code only and don't write new code from scratch .)

2. Write a function that recursively calculates the gcd of two numbers a and b using ONLY the Euclidean algorithm for gcd .

Steps of the Euclidean Algorithm:

- Start with two numbers a and b, where a>b.
- Divide a by b and find the remainder r.
- Replace a with b and b with r.
- Repeat the process until b becomes zero.
- When b is zero, the current value of a is the GCD of the original pair of numbers.

Example:

Input: a = 20, b = 28

Output: 4

Explanation: The factors of 20 are 1, 2, 4, 5, 10 and 20. The factors of 28 are 1, 2, 4, 7, 14 and 28. Among these factors, 1, 2 and 4 are the common factors of both 20 and 28. The greatest among the common factors is 4.

Input: a = 60, b = 36

Output: 12

**Input Format**

First line contains two space separated integers a and b .

**Constraints**

1 <= a ,b <= 10^5

**Output Format**

Output a single integer denoting the gcd of a and b

**Sample Input 0**

20 4

**Sample Output 0**

4

**Sample Input 1**

30 10

**Sample Output 1**

10

3. You and your friend want to name the new dog on the street. You will pick the firstname and he will pick the lastname. However, turns out that both of you have faulty vocabularies.

You only know certain English alphabets, and he knows certain other. Print all possible names that you and your friend can come up with given that you both use all the letters you know, exactly once, to form a name and you pick the first name and he picks the lastname.

Use recursion for printing all the names. You must print all the names in lexicographically increasing order.

**Input Format**

The first line contains a string of the alphabets known by you, all distinct. The second line contains a string of the alphabets known by him, all distinct. *The first and the last name may contain common letters. The letters are only distinct within the names themselves.

**Constraints**

1 <= len(firstname) <= 6 1 <= len(lastname) <= 6

**Output Format**

Output all the names in separate lines with a space in between the name and the surname. See the sample for a better understanding of the problem.