List Comprehensions

Let's learn about list comprehensions! You are given three integers X, Y and Z representing the dimensions of a cuboid along with an integer N. You have to print a list of all possible coordinates given by (i, j, k) on a 3D grid where the sum of $i + j + k$ is not equal to N. Here,
$0 \leq i \leq X; 0 \leq j \leq Y; 0 \leq k \leq Z$

Input Format

Four integers X, Y, Z and N each on four separate lines, respectively.

Constraints

Print the list in lexicographic increasing order.

Sample Input

```
1
1
1
2
```

Sample Output

```
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0], [1, 1, 1]]
```

Explanation

Concept

You have already used lists in previous hacks. List comprehensions are an elegant way to build a list without having to use different *for* loops to append values one by one. These examples might help.

The simplest form of a list comprehension is:

[expression-involving-loop-variable for loop-variable in sequence]

This will step over every element in a sequence, successively setting the loop-variable equal to every element one at a time. It will then build up a list by evaluating the expression-involving-loop-variable for each one. This eliminates the need to use lambda forms and generally produces a much more readable code than using *map()* and a more compact code than using a *for* loop.

```
>> ListOfNumbers = [ x for x in range(10) ] # List of integers from 0 to 9
>> ListOfNumbers
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List comprehensions can be nested where they take the following form:

[expression-involving-loop-variables for outer-loop-variable in outer-sequence for inner-loop-variable in inner-sequence]

This is equivalent to writing:

```
results = []
for outer_loop_variable in outer_sequence:
    for inner_loop_variable in inner_sequence:
        results.append( expression_involving_loop_variables )
```

The final form of list comprehension involves creating a list and filtering it similar to using the *filter()* method. The filtering form of list comprehension takes the following form:

[expression-involving-loop-variable for loop-variable in sequence if boolean-expression-involving-loop-variable]

This form is similar to the simple form of list comprehension, but it evaluates boolean-expression-involving-loop-variable for every item. It also only keeps those members for which the boolean expression is *True*.

```
>> ListOfThreeMultiples = [x for x in range(10) if x % 3 == 0] # Multiples of 3 below 10
>> ListOfThreeMultiples
[0, 3, 6, 9]
```