

# List Comprehensions



Lets learn about list comprehensions!. You are given three integers X, Y and Z denoting the dimensions of a Cuboid. You have to print a list of all possible coordinates on the three dimensional grid, such that at any point the sum  $X_i + Y_i + Z_i$  is not equal to N. If X = 2, then possible values of  $X_i$  can be 0, 1 and 2. The same applies to Y and Z.

## Input Format

Four integers X, Y, Z and N in four different lines.

## Output Format

You have to print the list, in 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]]
```

## Concept

You already have used list in some of the previous hacks. List comprehensions are an elegant way in which list can be built 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 of sequence, successively setting loop-variable equal to every element one at a time, and will then build up a list by evaluating expression-involving-loop-variable for each one. This eliminates the need to use lambda forms, and thus generally produces a much more readable code than using map() and a more compact code than using a for-loop.

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

List comprehensions can be nested, in which case they take on 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 similarly to using filter(). 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 and keeps only those members for which the boolean expression is True.

```
lis = [x for x in range(10) if x % 3 == 0] # Multiples of 3 below 10
```

```
lis
```

```
[0, 3, 6, 9]
```