ODOMETER PROBLEM

00000
00001
00002
..
00009
00010
....
09999
10000
...
99999

f(2 3 5 _ _, 3) → will print out all odometer setings where the first 3 digits are set to 2 3 5.

23500
23501
...
23599

2350_ → do this first f(2 3 5 0 _, 4)
2351_ → do this next f(2 3 5 1 _, 4)
2352_ → do this third f(2 3 5 2 _, 4)
…
2359_ → do this last f(2 3 5 9 _, 4)

```
printodom(odom, k):

    if k == len(odom):
        print(odom)
        return

    for i in range(10):
        odom[k] = i
        printodom(odom, k+1)
```

Imagine odometer with only 0s and 1s:

Odometer	Set
012	
000	{ }
001	{2}
010	{1}
011	{1,2}
100	{0}
101	{0,2}
110	{0,1}
111	{0,1,2}

Binary Representation of a number

000	0	{}
001	1	{0}
010	2	{1}
011	3	{1,0}
100	4	{2}
101	5	{2,0}

110	6	{2,1}
111	7	{2,1,0}

$1011001 = 1x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

Int x; // x = 5, how do I access the bits of x???

x & y → bitwise and (set intersection)

x = 1010	(10)
y = 1100	(12)
1000	(8)

X | y → bitwise or (set union)

x = 1010	(10)
y = 1100	(12)
1110	(14)

The value of n 1s: 1111 is simply $2^n - 1$.

X^y → bitwise xor (light switches, grading TF exams)

X = 1010 (10)
Y = 1100 (12)

0110 (6), 1 in xor means those bits are Different in x and y,

If x^y = z, then x^z = y and y^z = x.

How do I look at a single bit and determine if it's on or not?

a << b, this is left shift a by b bits.

1010 << 3 → 1010000
This is basically multiplying a by 2^b so long as there is no overflow.

1 << n → this equals 2^n

a >> b → this right shifts a by b bits, chopping off the last b bits

10101101**11101** >> 5 → 1010110

To look at bit i, do this:

```
if  (x & (1<<i)) != 0:
      // Bit i is on.
Else:
      .. Bit i is off.
```

X = 1011**1**00001, I = 4
(1<<4)= 00000 **1**0000

A common problem is given n items, say
0, 1, 2, …, n-1

List all orderings of those items.
0,1,2
0,2,1
1,0,2

1,2,0
2,0,1
2,1,0

Only difference between this and the odometer is no repeats...
We need to edit the odometer but build in a system to prevent repeats...

SKETCH OF ODOM CODE:

```
If k == length:
    // PROCESS
    Return

For i in range(choices):
    IF i has already been placed, skip it!
    Odom[k] = i
    Odometer(odom, k+1)
```

0 _ _ → need some way to skip filling in 0 here
0 1 _ → here skip BOTH 0 and 1.

Idea → add an used array, where used[i] = true if i has already placed in the array, and false if it hasn't.

DERANGEMENTS

0 1 2
1 2 0
2 0 1

For all I,

Perm[i] != i…no person gets their own hat.