Studying permutations of {1,2,3,4,5} -- list of permutations with their ranks

0 (1 2 3 4 5)	24 (2 1 3 4 5)	48 (3 1 2 4 5)	72 (4 1 2 3 5)	96 (5 1 2 3 4)
1 (1 2 3 5 4)	25 (2 1 3 5 4)	49 (3 1 2 5 4)	73 (4 1 2 5 3)	97 (5 1 2 4 3)
2 (1 2 4 3 5)	26 (2 1 4 3 5)	50 (3 1 4 2 5)	74 (4 1 3 2 5)	98 (5 1 3 2 4)
3 (1 2 4 5 3)	27 (2 1 4 5 3)	51 (3 1 4 5 2)	75 (4 1 3 5 2)	99 (5 1 3 4 2)
4 (1 2 5 3 4)	28 (2 1 5 3 4)	52 (3 1 5 2 4)	76 (4 1 5 2 3)	100 (5 1 4 2 3)
5 (1 2 5 4 3)	29 (2 1 5 4 3)	53 (3 1 5 4 2)	77 (4 1 5 3 2)	101 (5 1 4 3 2)
6 (1 3 2 4 5)	30 (2 3 1 4 5)	54 (3 2 1 4 5)	78 (4 2 1 3 5)	102 (5 2 1 3 4)
7 (1 3 2 5 4)	31 (2 3 1 5 4)	55 (3 2 1 5 4)	79 (4 2 1 5 3)	103 (5 2 1 4 3)
8 (1 3 4 2 5)	32 (2 3 4 1 5)	56 (3 2 4 1 5)	80 (4 2 3 1 5)	104 (5 2 3 1 4)
9 (1 3 4 5 2)	33 (2 3 4 5 1)	57 (3 2 4 5 1)	81 (4 2 3 5 1)	105 (5 2 3 4 1)
10 (1 3 5 2 4)	34 (2 3 5 1 4)	58 (3 2 5 1 4)	82 (4 2 5 1 3)	106 (5 2 4 1 3)
11 (1 3 5 4 2)	35 (2 3 5 4 1)	59 (3 2 5 4 1)	83 (4 2 5 3 1)	107 (5 2 4 3 1)
12 (1 4 2 3 5)	36 (2 4 1 3 5)	60 (3 4 1 2 5)	84 (4 3 1 2 5)	108 (5 3 1 2 4)
13 (1 4 2 5 3)	37 (2 4 1 5 3)	61 (3 4 1 5 2)	85 (4 3 1 5 2)	109 (5 3 1 4 2)
14 (1 4 3 2 5)	38 (2 4 3 1 5)	62 (3 4 2 1 5)	86 (4 3 2 1 5)	110 (5 3 2 1 4)
15 (1 4 3 5 2)	39 (2 4 3 5 1)	63 (3 4 2 5 1)	87 (4 3 2 5 1)	111 (5 3 2 4 1)
16 (1 4 5 2 3)	40 (2 4 5 1 3)	64 (3 4 5 1 2)	88 (4 3 5 1 2)	112 (5 3 4 1 2)
17 (1 4 5 3 2)	41 (2 4 5 3 1)	65 (3 4 5 2 1)	89 (4 3 5 2 1)	113 (5 3 4 2 1)
18 (1 5 2 3 4)	42 (2 5 1 3 4)	66 (3 5 1 2 4)	90 (4 5 1 2 3)	114 (5 4 1 2 3)
19 (1 5 2 4 3)	43 (2 5 1 4 3)	67 (3 5 1 4 2)	91 (4 5 1 3 2)	115 (5 4 1 3 2)
20 (1 5 3 2 4)	44 (2 5 3 1 4)	68 (3 5 2 1 4)	92 (4 5 2 1 3)	116 (5 4 2 1 3)
21 (1 5 3 4 2)	45 (2 5 3 4 1)	69 (3 5 2 4 1)	93 (4 5 2 3 1)	117 (5 4 2 3 1)
22 (1 5 4 2 3)	46 (2 5 4 1 3)	70 (3 5 4 1 2)	94 (4 5 3 1 2)	118 (5 4 3 1 2)
23 (1 5 4 3 2)	47 (2 5 4 3 1)	71 (3 5 4 2 1)	95 (4 5 3 2 1)	119 (5 4 3 2 1)

Generating permutations in lexicographical order

Permutations of
{1,2,3,4,5,6,7,8,9}

Permutation:
(5 1 8 3 9 7 6 4 2)
Next permutation:
(5 1 8 4 2 3 6 7 9)

Lexicographical order of permutations

Next permutation of
(5 1 8 3 9 7 6 4 2)

1.
Identify last increasing neighbour pair -- 3 and 9
(5 1 8 **3 9** 7 6 4 2)

2.
Swap 3 with the smallest value bigger than 3
to the right of 3:
(5 1 8 **4** 9 7 6 **3** 2)

3. Reverse the sequence to the right of 4
(= to the right of the original position of 3)
(5 1 8 4 **9 7 6 3 2**)

Generating permutations in lexicographical order

```
def nextperm( perm ):
    n = len(perm)
    # start at the last position
    j = n-1
    #find last ascending pair
    while True:
        if j == 0:
            return False # no next permutation
        if perm[j-1] > perm[j]:
            j -= 1
        else: break
    j1 = j-1 # remember  position
    # find smallest bigger element than perm[j1] to the right of j1
    while j < n and perm[j] > perm[j1]:  j += 1
    # index of that element:
    j -= 1
    perm[j], perm[j1] = perm[j1], perm[j] # swap
    # reverse sequence to the right of j1
    j1 += 1; j = n-1
    while j1 < j:
        perm[j], perm[j1] = perm[j1], perm[j]
        j1 += 1; j -= 1
    return True
```

Ranks of permutations

The rank of permutation (3,2,5,4,1) in the list of all permutations of {1,2,3,4,5}.

General strategy:

1. Note that the list of all permutations is divided into a number of blocks.

2. Establish the pattern by which the permutations are divided into blocks.

3. Note that this pattern has recursive character.

4. Using the established pattern, count (recursively) the number of blocks which precede the given permutation (3,2,5,4,1) in the list of all permutations . This number is equal to the rank of the subset.

24 (2 1 3 4 5)
25 (2 1 3 5 4)
26 (2 1 4 3 5)
27 (2 1 4 5 3)
28 (2 1 5 3 4)
29 (2 1 5 4 3)

Note the regular sizing of the blocks.

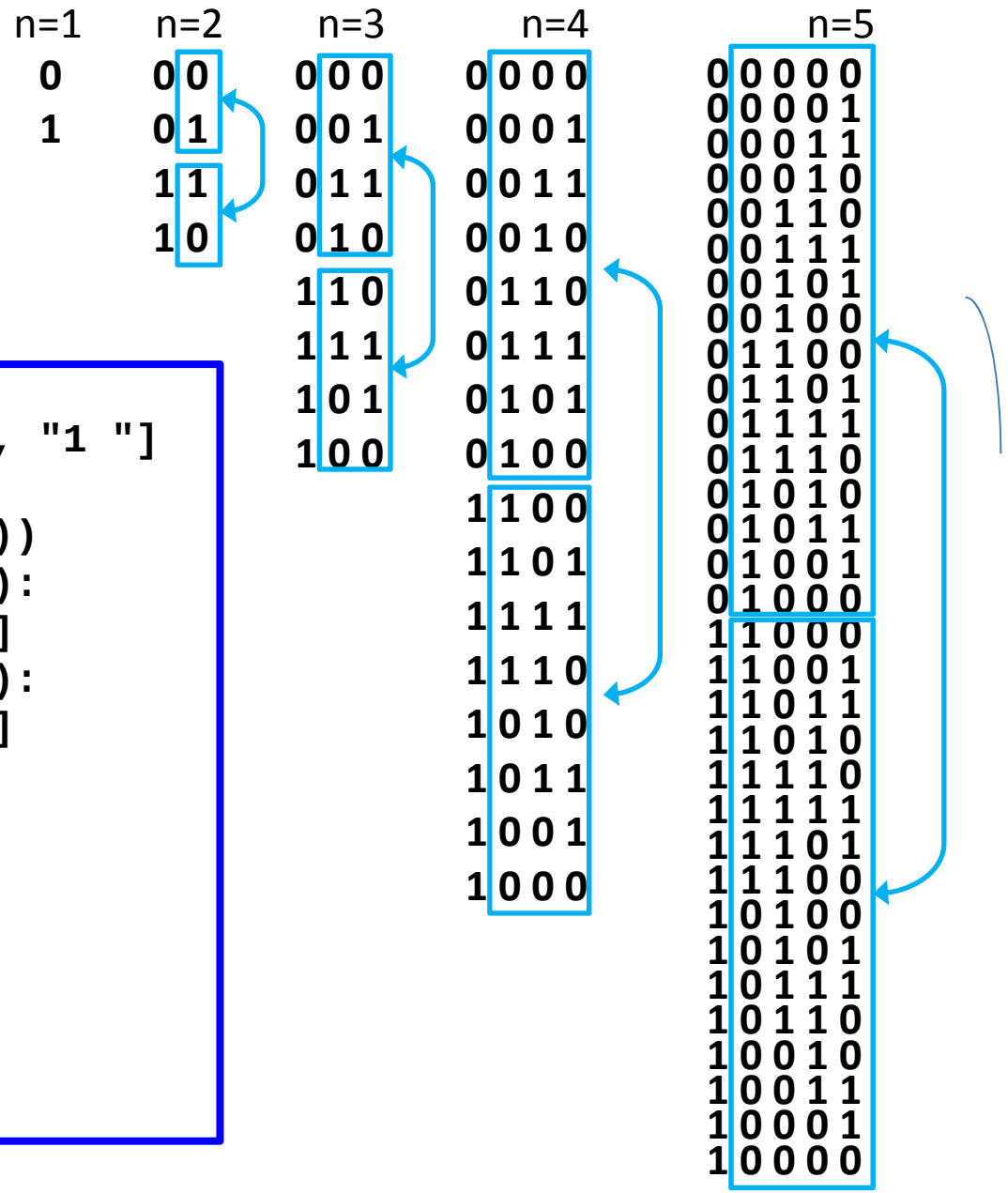
Ranks of permutations

```
def rankPermutation( perm ):
    n = len( perm )
    if n == 1: return 0

    rank = (perm[0]-1) * factorial(n-1)

    # consider the permutation without the 1st element
    perm1 = perm[1:]  # copy w/o perm[0]
    # "normalize" the resulting permutation
    # for recursive processing
    for j in range(len(perm1)):
        if perm1[j] > perm[0]:
            perm1[j] -= 1

    return rank + rankPermutation( perm1 )
```

Ranks of permutations

```
def unrankPerm( rank, permLen ):
    n = permLen # just synonym
    if n == 1: return [1]  # simplest possible permutation

    # count how many blocks of size n-1
    # would fit into list of all permutations
    # before the given rank
    blocksCount = rank // factorial(n-1)  # integer div

    # construct the first element of the permutation
    firstElem = blocksCount + 1

    # calculate remaining rank to feed into recursion
    rank = rank % factorial(n-1)

    # exploit recursion
    perm = unrankPerm( rank, n-1)

    # "fit" the returned permutation to current size n
    for j in range( len(perm)):
        if perm[j] >= firstElem:
            perm[j] += 1

    return [firstElem] + perm  # concatenate lists
```

Gray code examples




```
def grayCode( n ):
    if n == 1: return ["0 ", "1 "]
    gc0 = grayCode(n-1)
    gc1 = list(reversed(gc0))
    for i in range(len(gc0)):
        gc0[i] = "0 "+gc0[i]
    for i in range(len(gc1)):
        gc1[i] = "1 "+gc1[i]

    return gc0+gc1


for i in range (1,6):
    for z in grayCode( i ):
        print(z)
    print()
```