Analysis of Combinatorial Problems in the Python Programming Language

Alimova Rayhon Abdug'aforovna — Student of the Faculty of Information Technologies of Termiz State University

O`rolova Munira Ahmad qizi — Student of the Faculty of Information Technologies of Termiz State University

ABSTRACT

This article describes combinatorics and its combinations, and a program in the python programming language that displays the result.

Keywords: combinatorics, combinatorics problems, basic combinations, ordered set, python, permutations, non-repeated permutations, non-repeated groupings, repeated permutations.

Problems related to finding different combinations of elements and their number are called combinatorics problems. Such problems are studied in combinatorics, a branch of mathematics. Combinatorics mainly emerged as an independent science in the 17th-19th centuries, and scientists such as B. Pascal, P. Fermat, G. Leibniz, Y. Bernoulli, and L. Euler contributed greatly to its development. In the branch of mathematics called combinatorial analysis, combinatorial mathematics, combination theory, in short, combinatorics, a finite set or a set satisfying the condition of finiteness in a certain sense (it does not matter what the elements of this set are: letters, numbers, events, some objects, etc.) issues related to the division into parts, their placement and interposition, i.e., combinations, combinatorial structures are studied. Currently, combinatorics information is used in various fields of human activity. In particular, specialists dealing with mathematics, chemistry, physics, biology, linguistics, information technologies and other fields face various problems of combinatorics. Combinatorics studies basic combinations called permutations, permutations, and groupings.

If the elements of a finite set X are numbered in some way, we call it an ordered set: X = {x_1, x_2, …, x_(n)}. Unlike the concept of a tuple, there are no equals among the elements of an ordered set. For example, the tuple (2,3,2,4,5) is not an ordered set, and (2,3,4,5) is an ordered set. All possible combinations that differ from each other in the arrangement of elements are called permutations of n elements. In fact, the phrase "permutation" means the action of changing the positions of the elements of the set, but here we use it as the structure resulting from this action. We use this expression in its original meaning. The number of all possible permutations of n elements is defined as P_n (P is the first letter of the French word permutation).

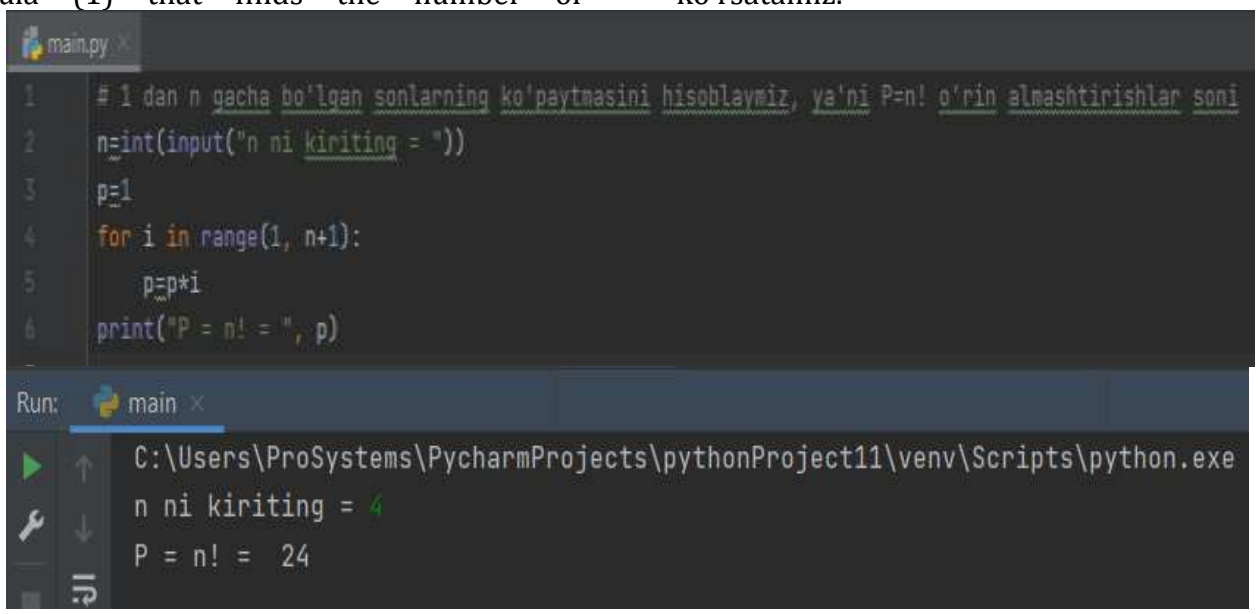
$$P_n = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1 = n! \qquad (1)$$

Example 1: From the elements of the set {2, 5, 7, 8}, it is possible to make several 4-digit numbers without repeated numbers.

Solution: 4 numbers, i.e. n=4. Then, based on the formula (1) above, P_4 = 4! = 4·3·2·1=24, so 24 numbers can be made.

Let's consider the calculation of the formula (1) that finds the number of substitutions in the Python programming language.

Yuqoridagi 1 – misol shartiga asosan n ga 4 ni kiritamiz va tuzilgan dasturning to'g'riligini ko'rsatamiz.



The result is 24, which means that the program code is working correctly. Even if we enter an arbitrary positive integer to n, the compiled program can easily and quickly calculate the number of permutations. This is also very convenient for all problems of combinatorics that require the calculation of the number of permutations, because sometimes large numbers can appear in the place of n, and it is difficult to calculate it manually.

Now let's consider the problem of how many ordered sets of m elements can be formed from the elements of the set X of n elements. Here, the creation of an ordered set with m elements is completed by taking m elements. To find the number of such ordered sets, it is enough to multiply the numbers m by n, n-1, n-2, ... , n-m+1. So, the number of ordered sets with m elements in X set is equal to $A_n^m = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-m+1)$. Such ordered sets are called unique arrangements of n elements from m. By multiplying and dividing the expression of A_n^m by $1 \cdot 2 \cdot \ldots \cdot (n-m)$, we can change its appearance:

$$A_n^m = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-m+1) \cdot (n-m) \cdot \ldots \cdot 2 \cdot 1}{1 \cdot 2 \cdot \ldots \cdot (n-m)} = \frac{n!}{(n-m)!} (2)$$

Two different placements with the same number of elements differ from each other by their elements or by the arrangement of these elements. For example, there are six permutations of three elements A, B, C with two elements: AB, AC, BC, BA, CA, CB. In addition, it is also clear that n ≥ m for m permutations of n elements.

Example 2: Let the group consist of 25 students. In this group, it is necessary to elect a group leader, an assistant group leader and a union representative for the group. Assuming that each student performs only one of these tasks, what are the possible outcomes of the election?

Therefore, we solve the problem according to formula (2). Since n=25, m=3, we found the number of possibilities for the following election results $A_{25}^3 = \frac{25!}{(25-3)!} = 25 \cdot 24 \cdot 23 = 13800$

Solution: Here, it is necessary to determine the number of ordered 3-element subsets of the 25-element set of students (group leader, assistant group leader, and representative of the union for the group). This means finding the number of permutations of 3 out of 25 elements. Therefore, we solve the problem according to formula (2). Since n=25, m=3, we found the number of possibilities for the following

election results $A_{25}^{3} = \frac{25!}{(25-3)!} = 25 \cdot 24 \cdot 23 =$ 13800

Let's consider the calculation of the formula (2) that finds the number of unique placements in the Python programming language.

```
main.py
# 1 dan n gacha bo'lgan sonlarning ko'paytmasini hisoblaymiz, ya'ni n!
n=int(input("n ni kiriting = "))
m=int(input("m ni kiriting = "))
p=1
for i in range(1, n+1):
    p=p*i

# 1 dan n-m gacha bo'lgan sonlarning ko'paytmasini hisoblaymiz, ya'ni (n-m)!
k=1
for i in range(1, n-m+1):
    k=k*i

print("Takrorsiz o'rinlashtirishlar soni  n!/(n-m)! = ", p/k)
```

To confirm that this is correct, if we enter the values n=25 and m=3 entered in the above example 2 into the written program and run it, we will get the following result

```
Run:  main
C:\Users\ProSystems\PycharmProjects\pythonProject11\venv\Scripts\python.
n ni kiriting = 25
m ni kiriting = 3
Takrorsiz o'rinlashtirishlar soni  n!/(n-m)! =  13800.0
```

So, even if we enter arbitrary positive integers into n and m (n≥m), the program will easily and quickly extract the number of unique placements. This is also very convenient for all problems of combinatorics that require the calculation of the number of unique permutations.

Now, how many subsets of m elements each can be formed from n elements of X? let's look at the issue. Such subsets are called unique groupings of n elements and m. Their number is determined by C_n^m. It is also possible to specify the number of groupings in the form $\binom{m}{n}$ or $\binom{n}{m}$. It is below

$$C_n^m = \frac{n!}{m!\,(n-m)!} \quad (3)$$

determined by the formula.

Example 3: How many groups of 3 workers can be formed from a group of 15?

Solution: We solve this grouping problem based on the formula (3), we take n=15 and m=3. Then $C_{15}^{3} = \frac{15!}{3!(15-3)!} = 455$.

We will write the general program code for the unique grouping formula in the Python programming language. The reason why it is called "general" is that it is appropriate for all problems of combinatorics whose solution is found on the basis of this formula. That is, the program code works correctly and can find a solution even with arbitrary (n≥m) values of n and m based on the condition of the problem.

```python
# 1 dan n gacha bo'lgan sonlarning ko'paytmasini hisoblaymiz, ya'ni n!
n=int(input("n ni kiriting = "))
m=int(input("m ni kiriting = "))
s=1
for i in range(1, n+1):
    s=s*i

# 1 dan n-m gacha bo'lgan sonlarning ko'paytmasini hisoblaymiz, ya'ni (n-m)!
k=1
for i in range(1, n-m+1):
    k=k*i

# 1 dan m gacha bo'lgan sonlarning ko'paytmasini hisoblaymiz, ya'ni m!
p=1
for i in range(1, m+1):
    p=p*i

print("Takrorsiz guruhlashlar soni  n!/(m!*(n-m)!) = ", s/(p*k))
```

Let's check this with values n=15, m=3 in example 3:

```
C:\Users\ProSystems\PycharmProjects\pythonProject11\venv\Scripts\python.
n ni kiriting = 15
m ni kiriting = 3
Takrorsiz guruhlashlar soni  n!/(m!*(n-m)!) =  455.0
```

In addition to these, for the number of repeated substitutions

$$C_n(n_1, n_2, n_3, \dots, n_k) = \frac{n!}{n_1! \cdot n_2! \cdot n_3! \cdot \dots \cdot n_k!} \quad (4)$$

the formula is appropriate, where $n_1 + n_2 + n_3 + \cdots + n_k = n$ is the number of elements, k is the number of types.

Example 4: Create all repeated permutations for a tuple consisting of two a's, one b's and two s's.

Solution: since $k = 3, n = 5, n_1 = 2, n_2 = 1, n_3 = 2$, we find the number of repeated permutations based on formula (4): $C_5(2, 1, 2) = \frac{5!}{2! \cdot 1! \cdot 2!} = 30$ will be.

Based on the condition of example 4 above, we will create a python program:

```python
n=int(input("n ni kiriting = "))
n1=int(input("n1 ni kiriting = "))
n2=int(input("n2 ni kiriting = "))
n3=int(input("n3 ni kiriting = "))
s=1
for i in range(1, n+1):
    s=s*i
k=1
for i in range(1, n1+1):
    k=k*i
l=1
for i in range(1, n2+1):
    l=l*i
m=1
for i in range(1, n3+1):
    m=m*i
print("Takrorli o'rin almashtirishlar soni  n!/(n1!*n2!*n3!) =",  s/(k*l*m))
```

If we enter the values $n = 5, n_1 = 2, n_2 = 1, n_3 = 2$, the following window will appear:

```
C:\Users\ProSystems\PycharmProjects\pythonProject11\venv\Scripts\python.
n ni kiriting = 5
n1 ni kiriting = 2
n2 ni kiriting = 1
n3 ni kiriting = 2
Takrorli o'rin almashtirishlar soni  n!/(n1!*n2!*n3!) = 30.0
```

Conclusion: often, in the process of studying the properties of things and events, we compare the elements of the studied object with each other, try to make different conclusions by looking at them together or by dividing them into pieces with elements. We are based on the concepts, formulas and properties of combinatorics. The information, solution examples and their analysis in the python programming language presented in this article are also important for those interested in mathematics and programming.

References:

1. Khalikulov S. I., Kuljanov O'., Ostonov Q. Elements of combinatorics. Methodological manual - Samarkand: SamDU publication, 2020.
2. Reingold, Yu. Nivergelyp, N. Deo. combinatorial algorithms. Theory and practice. M., Mir, 1980.
3. L. Erosh, Discrete Mathematics. Combinatorics: Textbook. SPb GUAP.SPb., 2001.
4. F. Rajabov. S. Masharipov. R. Madrahimov. "Higher Mathematics". Tashkent: "Turon - Iqbal", 2007.
5. N. Torayev, I. Azizov, S. Otakulov. "Combinatorics and graph theory" Tashkent - "Ilm zia" - 2009.