Combo

v0.1.0 2025-10-07 GPL-3.0-or-later

An importable library for common combinatorial operations

Michele Dusi
@micheledusi

combo is a library for **combinatorial operations** in Typst[1]. Use it to count the number of permutations, get every combination of items in a list, or other similar tasks.

Table of Contents

[1] https://typst.app/

Part I
Documentation

I.1 Combinations

`#combo.count-combinations` `#combo.count-combinations-no-repetition` `#combo.count-combinations-with-repetition` `#combo.get-combinations` `#combo.get-combinations-no-repetition` `#combo.get-combinations-with-repetition`

`#combo.count-combinations((items), (k), (repetition): false)` → `int`

Computes the **number of combinations** of n elements taken k at a time. The order of elements does not matter, and each element can be chosen only once unless repetition is allowed (by setting `repetition` to `true`).

- If `repetition` is `false`, the number is computed using the **binomial coefficient formula** (also known as *"n choose k"*) implemented as the `binom` function in the `calc` module:

$$C(n, k) = \frac{n!}{k! \cdot (n-k)!} = \text{binom}(n, k)$$

- If `repetition` is `true`, the number is computed using the formula for **combinations with repetition**:

$$C(n+k-1, k) = \text{binom}(n+k-1, k)$$

The returned number is given as an **integer**.

Argument

`(items)` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an array, its length is used. In the case of an integer, it must be a non-negative integer.

Argument

`(k)` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

Argument

`(repetition): false`

Whether to allow repetition of elements in the combinations. If `true`, elements can be chosen multiple times. If `false`, each element can be chosen only once. Default is `false`.

`#combo.count-combinations-no-repetition((items), (k)) → int`

Argument

`(items)` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an array, its length is used. In the case of an integer, it must be a non-negative integer.

Argument

`(k)` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

`#combo.count-combinations-with-repetition((items), (k)) → int`

Counts the **number of combinations with repetition**. Returns the number of possible combinations when choosing k items from n distinct items, allowing repetitions.

Argument

`(items)` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an array, its length is used. In the case of an integer, it must be a non-negative integer.

Argument

`(k)` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

`#combo.get-combinations((items), (k), (repetition): false) → array(array(int | any))`

Computes the **list of combinations** of indices for choosing k elements from a set of n elements. The order of elements does not matter, and each element can be chosen only once unless repetition is allowed (by setting `repetition` to `true`). The result is a list of tuples, where each tuple contains the indices of the chosen elements. Indices are zero-based, meaning they start from 0 up to $n - 1$.

- If `repetition` is `false`, combinations are generated without allowing any element to be chosen more than once.
- If `repetition` is `true`, combinations are generated allowing elements to be chosen multiple times.

The function returns an empty list if k is greater than n when `repetition` is `false`, or if either n or k is negative. If k is 0, the function returns a list containing an empty tuple, representing the single combination of choosing nothing.

Argument

`⟨items⟩` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an integer, it is interpreted as the total number of distinct items from 0 to $n - 1$.

Argument

`⟨k⟩` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

Argument

`⟨repetition⟩: false`

Whether to allow repetition of elements in the combinations. If `true`, elements can be chosen multiple times. If `false`, each element can be chosen only once. Default is `false`.

`#combo.get-combinations-no-repetition(⟨items⟩, ⟨k⟩)` → `array(array(int | any))`

Argument

`⟨items⟩` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an integer, it is interpreted as the total number of distinct items from 0 to $n - 1$.

Argument

`⟨k⟩` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

`#combo.get-combinations-with-repetition(⟨items⟩, ⟨k⟩)` → `array(array(int | any))`

Argument

`⟨items⟩` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an integer, it is interpreted as the total number of distinct items from 0 to $n - 1$.

Argument

`⟨k⟩` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

I.2 Permutations

#combo.count-permutations(⟨items⟩, ⟨k⟩: auto, ⟨repetition⟩: false) → int

Computes the **number of permutations** of n elements taken k at a time. The order of elements matters, and each element can be chosen only once unless repetition is allowed (by setting `repetition` to `true`).

- If `repetition` is `false`, the number is computed using the formula for **permutations without repetition** implemented as the `perm` function in the `calc` module:

$$P(n,k) = \frac{n!}{(n-k)!} = \text{perm}(n,k)$$

- If `repetition` is `true`, the number is computed using the formula for **permutations with repetition**:

$$n^k = \text{pow}(n,k)$$

The returned number is given as an **integer**.

Argument

⟨items⟩ — int | array

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an array, its length is used. In the case of an integer, it must be a non-negative integer.

Argument

⟨k⟩: auto — int

The number of items to choose, among the n distinct items. Must be a non-negative integer.

Argument

⟨repetition⟩: false — bool

Whether to allow repetition of elements in the permutations. If `true`, elements can be chosen multiple times. If `false`, each element can be chosen only once. Default is `false`.

#combo.count-permutations-no-repetition(⟨items⟩, ⟨k⟩: auto) → int

Calculates the number of permutations without repetition. Returns the number of possible permutations when choosing k items from n distinct items, without allowing repetitions.

Argument

⟨items⟩ — int | array

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an array, its length is used. In the case of an integer, it must be a non-negative integer.

Argument

`(k): auto` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

`#combo.count-permutations-with-repetition((items), (k): auto)` → `int`

Counts the **number of permutations with repetition**. Returns the number of possible permutations when choosing k items from n distinct items, allowing repetitions.

Argument

`(items)` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an array, its length is used. In the case of an integer, it must be a non-negative integer.

Argument

`(k): auto` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

`#combo.get-permutations((items), (k): auto, (repetition): false)` → `array(array(int | any))`

Generates all permutations of n elements taken k at a time. The order of elements matters. By default, each element can be chosen only once, but this can be changed by setting `repetition` to `true`. The list of permutations is represented as an array of tuples, where each tuple contains the indices of the chosen elements. Indices are zero-based, meaning they start from 0 up to $n - 1$. If k is not provided or set to `auto`, it defaults to n, generating permutations that use all elements. The function returns an empty list if k is greater than n when `repetition` is `false`, or if either n or k is negative. If k is 0, the function returns a list containing an empty tuple, representing the single permutation of choosing nothing.

- If `repetition` is `false`, permutations are generated without allowing any element to be chosen more than once.
- If `repetition` is `true`, permutations are generated allowing elements to be chosen multiple times.

Argument

`(items)` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an integer, it is interpreted as the total number of distinct items from 0 to $n - 1$.

Argument

`(k): auto` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

Argument

`(repetition): false` `bool`

Whether to allow repetition of elements in the permutations. If `true`, elements can be chosen multiple times. If `false`, each element can be chosen only once. Default is `false`.

`#combo.get-permutations-no-repetition((items), (k): auto)` → `array(array(int | any))`

Argument

`(items)` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an integer, it is interpreted as the total number of distinct items from 0 to $n - 1$.

Argument

`(k): auto` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

`#combo.get-permutations-with-repetition((items), (k): auto)` → `array(array(int | any))`

Generates all **permutations with repetition** of n elements taken k at a time. The order of elements matters, and each element can be chosen multiple times. The list of permutations is represented as an array of tuples, where each tuple contains the indices of the chosen elements. Indices are zero-based, meaning they start from 0 up to $n - 1$. If k is not provided or set to `auto`, it defaults to n, generating permutations that use all elements. The function returns an empty list if either n or k is negative.

Argument

`(items)` `int` | `array`

The array of distinct items, or the total number of distinct items. Both an array and an integer are accepted. In the case of an integer, it is interpreted as the total number of distinct items from 0 to $n - 1$.

Argument

`(k): auto` `int`

The number of items to choose, among the n distinct items. Must be a non-negative integer.

I.3 Auxiliary functions

`#combo.cartesian-product`

`#combo.cartesian-product(..(arrays)) → array(array(any))`

Computes the Cartesian product of a list of arrays. The Cartesian product is the set of all ordered pairs (or tuples) that can be formed by taking one element from each array. The result is a list of tuples, where each tuple contains one element from each array. If any of the arrays is empty, the result will be an empty set.

Argument

`..(arrays)` `array(array(any))`

A list of arrays to compute the Cartesian product. Each array can contain any type of elements.

Credits

This package is created by **Michele Dusi**[2] and is licensed under the GNU General Public License v3.0[3].

This project owes a lot to the creators of these **Typst packages**, whose work made COMBO possible:

- MANTYS[4] and TIDY[5] for documentation.

Special thanks to everyone involved in the development of the Typst[6] language and engine, whose efforts made the entire ecosystem possible.

Contributions are welcome!

Found a bug, have an idea, or want to contribute? Feel free to open an **issue** or **pull request** on the *GitHub* repository (micheledusi/Combo[7]).

Made something cool with COMBO? Let me know — I'd love to feature your work!

[2] https://github.com/micheledusi
[3] https://www.gnu.org/licenses/gpl-3.0.en.html
[4] https://typst.app/universe/package/mantys
[5] https://typst.app/universe/package/tidy
[6] https://typst.app/about/
[7] https://github.com/micheledusi/Combo

Part II

Index

C

G