# `combinat` — library for combinatorics

## Table of contents

## Introduction

The `combinat` library provides algorithms from some areas of combinatorics.

The package functions are called using the package name `combinat` and the name of the function. E.g., use

```
>> combinat::bell(5)
```

to compute the 5-th bell number. This mechanism avoids naming conflicts with other library functions. If this is found to be inconvenient, then the routines of the `combinat` package may be exported via `export`. E.g., after calling

```
>> export(combinat, bell)
```

the function `combinat::bell` may be called directly:

```
>> bell(5)
```

All routines of the `combinat` package are exported simultaneously by

```
>> export(combinat)
```

The functions available in the `combinat` library can be listed using

```
>> info(combinat)
```

## `combinat::bell` – **Computing Bell numbers**

`combinat::bell`(n) computes the n-th Bell number.

**Call(s):**

- `combinat::bell(n)`
- `combinat::bell(expression)`

**Parameters:**

| | | |
|---|---|---|
| `n` | — | nonnegative integer |
| `expression` | — | An expression of type `Type::Arithmetical` which must be a nonnegative integer if it is a number. |

**Return Value:** A positive integer value if `n` was a nonnegative integer. Otherwise `combinat::bell` returns the unevaluated function call.

---

**Details:**

- The n-th bell number is defined by the exponential generating function:

$$e^{e^x-1} = \sum_{n=0}^{\infty} \frac{bell(n)}{n!} x^n$$

  Often another definition is used. The n-th bell number is the number of different ways of partitioning the set $\{1, 2, \ldots, n\}$ into disjoint nonempty subsets, and bell(0) is defined to be 1.

- Bell numbers are computed using the formula:

$$\begin{aligned} bell(0) &= 1 \\ bell(n+1) &= \sum_{i=0}^{n} \binom{n}{i} bell(i) \qquad \text{for } n > 0 \end{aligned}$$

---

**Example 1.**

```
>> combinat::bell(3)

                                5
```

This means that you can partition the set $\{1, 2, 3\}$ into disjoint subsets in 5 different ways. These are $\{\{1, 2, 3\}\}$, $\{\{1\}, \{2, 3\}\}$, $\{\{2\}, \{1, 3\}\}$, $\{\{3\}, \{1, 2\}\}$, and $\{\{1\}, \{2\}, \{3\}\}$. Or, that you can write $105 = 3 * 5 * 7$ as 5 different products. These are $105 = 3 * 5 * 7 = 15 * 7 = 21 * 5 = 3 * 35 = 105 * 1$.

**Example 2.** If one uses a wrong argument, an error message is returned

```
>> combinat::bell(3.4)

 Error: Nonnegative integer expected [combinat::bell]
```

**Example 3.** One can see why it is useful to return the unevaluated function call.

```
>> a:=combinat::bell(x)

                        combinat::bell(x)

>> x :=4

                                4

>> a ; delete a:

                               15
```

**Changes:**

- In older MuPAD versions `combinat::bell` returned 1 for a negative integer. Now it returns an error message if it gets a negative integer as an argument.

---

## `combinat::cartesian` – Cartesian product of sets

`combinat::cartesian(set1, set2, ..., setN)` computes the cartesian product of the given sets `set1`, `set2`, ..., `setN`.

For every positive integer $n$, the set $\{1, \ldots, n\}$ may be denoted by `n`, and `0` may be written instead of the empty set.

**Call(s):**

- `combinat::cartesian(set1, set2, ..., setN)`

**Parameters:**

`set1, set2, ..., setN` — Sets of domain type `DOM_SET`, or nonnegative integers.

**Return Value:** A set of domain type `DOM_SET` containing $N$-tuples of domain type `DOM_LIST`, where $N$ is the number of arguments.

---

**Details:**

- The cartesian product of the given sets `set1`, `set2` is the set $set1 \times set2 \times \ldots \times setN$ of all $N$-tuples $[x1, x2, \ldots, xN]$ with $x_n \in setn$, $1 \leq n \leq N$.
- `combinat::cartesian`() is not commutative, as demonstrated in example 3.

---

**Example 1.** Which cards exist, if you have the following suits and numbers available?

```
>> combinat::cartesian({Diamondsuit,Heartsuit,Spadesuit,Clubsuit},{7,8,9,10
```

```
 {[Clubsuit, 7], [Clubsuit, 8], [Clubsuit, 9], [Clubsuit, 10],

    [Spadesuit, 7], [Spadesuit, 8], [Spadesuit, 9],

    [Spadesuit, 10], [Heartsuit, 7], [Heartsuit, 8],

    [Heartsuit, 9], [Heartsuit, 10], [Diamondsuit, 7],

    [Diamondsuit, 8], [Diamondsuit, 9], [Diamondsuit, 10]}
```

**Example 2.** The same as above, but with other numbers:

```
>> combinat::cartesian({Diamondsuit,Heartsuit,Spadesuit,Clubsuit},3)
```

```
 {[Clubsuit, 1], [Clubsuit, 2], [Clubsuit, 3], [Spadesuit, 1],

    [Spadesuit, 2], [Spadesuit, 3], [Heartsuit, 1],

    [Heartsuit, 2], [Heartsuit, 3], [Diamondsuit, 1],

    [Diamondsuit, 2], [Diamondsuit, 3]}
```

**Example 3.** The cartesian product isn't commutative:

```
>> combinat::cartesian({Diamondsuit},2); combinat::cartesian(2,{Diamondsuit
```

```
              {[Diamondsuit, 1], [Diamondsuit, 2]}

              {[1, Diamondsuit], [2, Diamondsuit]}
```

**Changes:**

- No changes.

---

## `combinat::choose` – Computes all k-subsets of a given set

`combinat::choose(set,k)` computes all `k`-subsets of the given set `set`

`combinat::choose(N,k)` computes all `k`-subsets of the set `setN` where `setN` `=` $\{1, 2, \ldots, N\}$.

**Call(s):**

- `combinat::choose(set,k)`
- `combinat::choose(N,k)`

**Parameters:**

`set` — a set of domain type `DOM_SET`
`k` — a nonnegative integer
`N` — a nonnegative integer

**Return Value:** `combinat::choose` returns an expression sequence, consisting of the computed subsets.

---

**Example 1.** Compute all the subsets of $\{a, b, c, d, e\}$ containing 3 elements

```
>> combinat::choose({a,b,c,d,e},3)

 {c, d, e}, {b, d, e}, {a, d, e}, {b, c, e}, {a, c, e},

    {a, b, e}, {b, c, d}, {a, c, d}, {a, b, d}, {a, b, c}
```

**Example 2.** Compute all the subsets of $\{1, 2, 3\}$ containing 2 elements

```
>> combinat::choose(3,2)

                    {2, 3}, {1, 3}, {1, 2}
```

**Example 3.** It's not a good idea to compute the subsets containing $-1$ element

```
>> combinat::choose({a,3},-1)

 Error: Second argument must be a nonnegative integer [combinat\
 ::choose]
```

**Changes:**

- No changes.

---

## `combinat::composition` – $k$-composition of an integer

`combinat::composition` computes a list of all distinct ordered $k$-tupels $(k_1, \ldots, k_n)$ such that $\sum_{i=1}^{k} n_i = n$ and $n_i \geq 1, i = 1 \ldots k$.

**Call(s):**

- `combinat::composition(n,k)`

**Parameters:**

`n, k` — integer

**Return Value:** A list of type `DOM_LIST` containing every computed $k$-tupel also as a list of type `DOM_LIST`. If there exist no $k$-tupel the empty list is returned.

---

**Details:**

- `combinat::composition(n, k)` returns an empty list if $n < 1$ or $k < 1$ or $n < k$.

---

**Example 1.** How can one write 5 as a sum of two other positive integers?

```
>> combinat::composition(5,2)

                [[1, 4], [2, 3], [3, 2], [4, 1]]
```

**Example 2.** There is no way to write 2 as the sum of 5 positive integers.

```
>> combinat::composition(2,5)

                                []
```

**Example 3.** `combinat::composition` does not handle symbolic expressions.

```
>> combinat::composition(xx,2)

 Error: arguments must be integers [combinat::composition]
```

**Changes:**

- No changes.

---

## `combinat::modStirling` – modified Stirling numbers

`combinat::modStirling` computes the modified Stirling numbers.

**Call(s):**

- `combinat::modStirling(q, n, k)`

**Parameters:**

`q` — the argument: an integer
`n` — the number of variables: a nonnegative integer
`k` — the degree: a nonnegative integer

**Return Value:** a positive integer.

---

**Details:**

- `combinat::modStirling(q,n,k)` takes the elementary symmetric polynomial in `n` variables of degree `k` and evaluates it for the values `q+1`, . . . , `q+n`. Note that `k` must not be greater then `n`.

---

**Example 1.**

```
>> combinat::modStirling(2,4,2)

                              119
```

**Changes:**

- `combinat::modStirling` is a new function.

---

## `combinat::partitions` – $n$-th partitions number

`combinat::partitions(n)` returns the number of partitions of the non-negative integer `n`.

**Call(s):**

- `combinat::partitions(n)`

**Parameters:**

`n` — a nonnegative integer

**Return Value:** The number of partitions as a positive integer.

---

**Details:**

- The number of partitions of the nonnegative integer `n` is the number of representations of `n` as $n = \sum_{i=1}^{k} n_i$, $n_i \geq 1, i = 1 \ldots k$. By definition `combinat::partitions(0)` is 1.
- For small `n` Euler's pentagonal formula is used to compute `combinat::partitions(n)`.

$$p(n) + \sum_{k=1}^{\infty} -1^k (p(n - w(k)) + p(n - w(-k))) = 0, \text{ where } w(k) = (3 * k^2 + k)/2$$

For large `n` the Hardy-Ramanujan-Rademacher formula is used.

---

**Example 1.** We can write 3 in 3 different ways as a sum of nonnegative integers. They are $3 = 1 + 1 + 1 = 1 + 2 = 3$.

```
>> combinat::partitions(3)

                               3
```

**Example 2.** The number of partitions of $n$ grows very rapidly for larger $n$.

```
>> combinat::partitions(111)

                           679903203
```

**Example 3.** A negative number cannot be written as a sum of positive integers.

```
>> combinat::partitions(-3)

 Error: Argument must be a nonnegative integer [combinat::parti\
 tions]
```

**Further Documentation:** G. Andrews, The Theory of Partitions, Addison-Wesley, 1976

**Changes:**

- No changes.

---

## `combinat::permute` – permutations of a list

`combinat::permute`(list) computes all the reorderings of the given list `list`.

`combinat::permute`(n) computes all the reorderings of the list $[1, 2, \ldots, n]$.

**Call(s):**

- `combinat::permute(n)`
- `combinat::permute(list)`
- `combinat::permute(list,`*`Duplicate`*`)`

**Parameters:**

| | |
|---|---|
| `n` | — a nonnegative integer |
| `list` | — a list |

**Options:**

| | |
|---|---|
| *`Duplicate`* | — The result may contain identical lists if there are duplicates in the given list `list`. |

**Return Value:** A list of type `DOM_LIST` containing every reordered list as an element.

---

**Details:**

- Without the option *`Duplicate`*, all lists in the result are distinct.

---

**Option <*`Duplicate`*>:**

- If the given list contains k elements, then the resulting list contains $k!$ elements, which do not have to be distinct. This means duplicates are not treated differently. Cf. examples 3 and 4.

---

**Example 1.** There are exactly two ways of ordering two elements.

```
>> combinat::permute([a,b])

                        [[a, b], [b, a]]
```

**Example 2.** An integer argument $n$ is equivalent to the list of the first $n$ integers.

```
>> combinat::permute(3)

 [[2, 3, 1], [3, 2, 1], [1, 3, 2], [3, 1, 2], [1, 2, 3],

    [2, 1, 3]]
```

**Example 3.** By default, one gets all *distinct* reorderings.

```
>> combinat::permute([a,a,b])

                [[a, b, a], [b, a, a], [a, a, b]]
```

**Example 4.** But if one wants to get a list with duplicated reordered entries, this is also possible.

```
>> combinat::permute([a,a,b],Duplicate)

 [[a, b, a], [b, a, a], [a, b, a], [b, a, a], [a, a, b],

    [a, a, b]]
```

**Example 5.** Sets are not allowed as an argument.

```
>> combinat::permute({3,4})

 Error: argument must be a list or a non-negative integer! [com\
 binat::permute]
```

**Changes:**

- In older MuPAD versions the option Duplicate was the default behaviour of the function `combinat::permute`().

---

## `combinat::powerset` – **power set of a set or list**

`combinat::powerset(set)` computes the powerset of the given set `set`, that is, the set of all subsets of `set`.

`combinat::powerset(list)` computes the powerset of the given list `list`, that is, the set of all sublists of `list`. In this context, lists are understood as multisets.

`combinat::powerset(n)` computes the powerset of the set $\{1, 2, \ldots, n\}$.

**Call(s):**

- `combinat::powerset(n)`
- `combinat::powerset(set)`
- `combinat::powerset(list)`

**Parameters:**

| | | |
|---|---|---|
| `n` | — | a nonnegative integer |
| `set` | — | a set of domain type `DOM_SET` |
| `list` | — | a list of domain type `DOM_LIST` |

**Return Value:** A set of domain type `DOM_SET` which contains the computed subsets.

**Overloadable by:** `set`

**Related Functions:** `combinat::choose`

---

**Details:**

- If the argument of `combinat::powerset` is a list, it is treated like a multiset. This means that sublists that contain the same elements the same number of times are treated as equal, even if the elements appear in a different order. Cf. Example 3.

---

**Example 1.**

```
>> combinat::powerset({a, b, c})

    {{}, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}}
```

**Example 2.**

```
>> combinat::powerset(3)

     {{}, {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}}
```

**Example 3.** Here you can see that lists are treated as multisets. There is no sublist $[2, 1]$ since it is identified with the list $[1, 2]$ which is in the powerset.

```
>> combinat::powerset([2, 1, 2])

            {[], [1], [2], [1, 2], [2, 2], [1, 2, 2]}
```

**Changes:**

- Extended to work on lists.

---

### `combinat::stirling1` – **Stirling numbers of the first kind**

`combinat::stirling1`(n,k) computes the Stirling numbers of the first kind.

**Call(s):**

- `combinat::stirling1(n,k)`

**Parameters:**

`n,k` — nonnegative integers

**Return Value:** an integer.

---

**Details:**

- Let $S(n, k)$ be the number of permutations of `n` symbols that have exactly `k` cycles. Then `combinat::stirling1`(n,k) computes $(-1)^{(n+k)}S(n, k)$.
- Let $S1(n, k)$ be the stirling number of the first kind, then we have:

$$\sum_{k=0}^{n} S1(n,k)x^k = x(x-1)\dots(x-n+1)$$

---

**Example 1.** Let us have a look what's the result of $x(x-1)(x-2)(x-3)(x-4)(x-5)$ written as a sum.

```
>> expand(x*(x-1)*(x-2)*(x-3)*(x-4)*(x-5))

              2                3       4       5    6
     274 x  - 120 x - 225 x  + 85 x  - 15 x  + x
```

Now let us "prove" the formula mentioned in the "Details" section by calculating the proper stirling numbers

```
>> combinat::stirling1(6,1);
   combinat::stirling1(6,2);
   combinat::stirling1(6,3);
   combinat::stirling1(6,4);
   combinat::stirling1(6,5);
   combinat::stirling1(6,6)

                          -120

                           274

                          -225

                           85

                           -15

                            1
```

**Example 2.**

```
>> combinat::stirling1(3,-1)

 Error:  Arguments must be nonnegative integers. [combinat::sti\
 rling1]
```

**Further Documentation:** J.J. Rotman, An Introduction to the Theory of Groups, 3rd Edition, Wm. C. Brown Publishers, Dubuque, 1988

**Changes:**

- No changes.

## `combinat::stirling2` – Stirling numbers of the second kind

`combinat::stirling2`(n,k) computes the Stirling numbers of the second kind.

**Call(s):**

- `combinat::stirling2(n,k)`

**Parameters:**

`n,k` — nonnegative integers

**Return Value:** a nonnegative integer.

---

**Details:**

- `combinat::stirling2`(n,k) computes the number of ways of partitioning a set of `n` elements into `k` non-empty subsets.
- `combinat::stirling2`(n,k) is calculated using the formula

$$stirling2(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n.$$

---

**Example 1.** One can partition the set $\{1,2,3\}$ into $\{1,2,3\} = \{1,2\} \cup \{3\} = \{1,3\} \cup \{2\} = \{2,3\} \cup \{1\}$

```
>> combinat::stirling2(3,2)

                              3
```

**Example 2.**

```
>> combinat::stirling2(3)

 Error: Two arguments expected. [combinat::stirling2]
```

**Changes:**

- No changes.