Contents

Chapter 1

Functions

1.1 combinatorial – combinatorial functions

1.1.1 binomial – binomial coefficient

binomial(`n`**:** ***integer*****,** `m`**:** ***integer*** **)** $\rightarrow$ ***integer***

Return the binomial coefficient for `n` and `m`. In other words, $\frac{\mathtt{n}!}{(\mathtt{n}-\mathtt{m})!\mathtt{m}!}$.

†For convenience, `binomial(n, n+i)` returns 0 for positive i, and `binomial(0,0)` returns 1.

`n` must be a positive integer and `m` must be a non-negative integer.

1.1.2 combinationIndexGenerator – iterator for combinations

combinationIndexGenerator(`n`**:** ***integer*****,** `m`**:** ***integer*** **)** $\rightarrow$ ***iterator***

Return an iterator which generates indices of `m` element subsets of `n` element set.

`combination_index_generator` is an alias of `combinationIndexGenerator`.

1.1.3 factorial – factorial

factorial(`n`**:** ***integer*** **)** $\rightarrow$ ***integer***

Return `n`! for non-negative integer `n`.

1.1.4 permutationGenerator – iterator for permutation

permutationGenerator(`n`: *integer*) → *iterator*

Generate all permutations of `n` elements as list iterator.

The number of generated list is `n`'s **factorial**, so be careful to use big `n`.

`permutation_generator` is an alias of `permutationGenerator`.

1.1.5 fallingfactorial – the falling factorial

fallingfactorial(`n`: *integer*, `m`: *integer*) → *integer*

Return the falling factorial; `n` to the `m` falling, i.e. $n(n-1)\cdots(n-m+1)$.

1.1.6 risingfactorial – the rising factorial

risingfactorial(`n`: *integer*, `m`: *integer*) → *integer*

Return the rising factorial; `n` to the `m` rising, i.e. $n(n+1)\cdots(n+m-1)$.

1.1.7 multinomial – the multinomial coefficient

multinomial(`n`: *integer*, parts: *list*) → *integer*

Return the multinomial coefficient.

parts must be a sequence of natural numbers and the sum of elements in parts should be equal to `n`.

1.1.8 bernoulli – the Bernoulli number

bernoulli(`n`: *integer*) → *Rational*

Return the `n`-th Bernoulli number.

1.1.9 catalan – the Catalan number

catalan(`n`: ***integer***) → ***integer***

Return the `n`-th Catalan number.

1.1.10 euler – the Euler number

euler(`n`: ***integer***) → ***integer***

Return the `n`-th Euler number.

1.1.11 bell – the Bell number

bell(`n`: ***integer***) → ***integer***

Return the `n`-th Bell number.

The Bell number b is defined by:

$$b(n) = \sum_{i=0}^{n} S(n, i),$$

where S denotes Stirling number of the second kind (**stirling2**).

1.1.12 stirling1 – Stirling number of the first kind

stirling1(`n`: ***integer***, `m`: ***integer***) → ***integer***

Return Stirling number of the first kind.

Let s denote the Stirling number and $(x)_n$ the falling factorial, then

$$(x)_n = \sum_{i=0}^{n} s(n,\ i)x^i.$$

s satisfies the recurrence relation:

$$s(n,\ m) = s(n-1,\ m-1) - (n-1)s(n-1,\ m)\ .$$

1.1.13 stirling2 – Stirling number of the second kind

stirling2(n: *integer*, m: *integer*) → *integer*

Return Stirling number of the second kind.

Let S denote the Stirling number, $(x)_i$ falling factorial, then:

$$x^n = \sum_{i=0}^{n} S(n,\ i)(x)_i$$

S satisfies:

$$S(n,\ m) = S(n-1,\ m-1) + mS(n-1,\ m)$$

1.1.14 partition_number – the number of partitions

partition_number(n: *integer*) → *integer*

Return the number of partitions of `n`.

1.1.15 partitionGenerator – iterator for partition

partitionGenerator(n: *integer*, maxi: *integer*=0) → *iterator*

Return an iterator which generates partitions of `n`.

If `maxi` is given, then summands are limited not to exceed `maxi`.
The number of partitions (given by **partition_number**) grows exponentially, so be careful to use big `n`.

`partition_generator` is an alias of `partitionGenerator`.

1.1.16 partition_conjugate – the conjugate of partition

partition_conjugate(partition: *tuple*) → *tuple*

Return the conjugate of `partition`.

Examples

```
>>> combinatorial.binomial(5, 2)
10L
>>> combinatorial.factorial(3)
6L
>>> combinatorial.fallingfactorial(7, 3) == 7 * 6 * 5
True
>>> combinatorial.risingfactorial(7, 3) == 7 * 8 * 9
True
>>> combinatorial.multinomial(7, [2, 2, 3])
210L
>>> for idx in combinatorial.combinationIndexGenerator(5, 3):
...     print idx
...
[0, 1, 2]
[0, 1, 3]
[0, 1, 4]
[0, 2, 3]
[0, 2, 4]
[0, 3, 4]
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]
>>> for part in combinatorial.partitionGenerator(5):
...     print part
...
(5,)
(4, 1)
(3, 2)
(3, 1, 1)
(2, 2, 1)
(2, 1, 1, 1)
(1, 1, 1, 1, 1)
>>> combinatorial.partition_number(5)
7
>>> def limited_summands(n, maxi):
...     "partition with limited number of summands"
...     for part in combinatorial.partitionGenerator(n, maxi):
...         yield combinatorial.partition_conjugate(part)
...
>>> for part in limited_summands(5, 3):
...     print part
...
(2, 2, 1)
```

```
(3, 1, 1)
(3, 2)
(4, 1)
(5,)
```

Bibliography