**itertools.product()**

This tool computes the cartesian product of input iterables.

It is equivalent to nested for-loops.

For example, product(A, B) returns the same as ((x,y) for x in A for y in B).

Sample Code

>>> from itertools import product

>>>

>>> print list(product([1,2,3],repeat = 2))

[(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)]

>>>

>>> print list(product([1,2,3],[3,4]))

[(1, 3), (1, 4), (2, 3), (2, 4), (3, 3), (3, 4)]

>>>

>>> A = [[1,2,3],[3,4,5]]

>>> print list(product(\*A))

[(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 3), (3, 4), (3, 5)]

>>>

>>> B = [[1,2,3],[3,4,5],[7,8]]

>>> print list(product(\*B))

[(1, 3, 7), (1, 3, 8), (1, 4, 7), (1, 4, 8), (1, 5, 7), (1, 5, 8), (2, 3, 7), (2, 3, 8), (2, 4, 7), (2, 4,

itertools.permutations()

36/115 challenges solved

Rank: 52422|Points: 505

Python

itertools.permutations(iterable[, r])

This tool returns successive length permutations of elements in an iterable.

If is not specified or is None, then defaults to the length of the iterable, and all possible full length permutations are generated.

Permutations are printed in a lexicographic sorted order. So, if the input iterable is sorted, the permutation tuples will be produced in a sorted order.

Sample Code

>>> from itertools import permutations

>>> print permutations(['1','2','3'])

<itertools.permutations object at 0x02A45210>

>>>

>>> print list(permutations(['1','2','3']))

[('1', '2', '3'), ('1', '3', '2'), ('2', '1', '3'), ('2', '3', '1'), ('3', '1', '2'), ('3', '2', '1')]

>>>

>>> print list(permutations(['1','2','3'],2))

[('1', '2'), ('1', '3'), ('2', '1'), ('2', '3'), ('3', '1'), ('3', '2')]

>>>

>>> print list(permutations('abc',3))

[('a', 'b', 'c'), ('a', 'c', 'b'), ('b', 'a', 'c'), ('b', 'c', 'a'), ('c', 'a', 'b'), ('c', 'b', 'a')]